



Politechnika Gdańska
WYDZIAŁ ELEKTRONIKI
TELEKOMUNIKACJI I
INFORMATYKI



Katedra: Architektury Systemów Komputerowych

Imię i nazwisko dyplomanta: Andrzej Jakowski

Nr albumu: 97015

Forma i poziom studiów: magisterskie

Kierunek studiów: informatyka

Praca dyplomowa magisterska

Temat pracy: Platforma do edycji ontologii z dostępem przez WWW

Kierujący pracą: dr inż. Paweł Czarnul

Zakres pracy:

W niniejszej pracy dyplomowej skupiono się nad stworzeniem warstwy prezentacji dla systemu składowania i pozyskiwania ontologii - OCS.

Część teoretyczna pracy dotyczy przedstawienia metod składowania ontologii, a także opisuje wizję Semantycznej Sieci Web i jej poszczególne fragmenty.

W części praktycznej pracy, rozpoznano i oceniono narzędzia pozwalające na edycję ontologii. W oparciu o specyfikację wymagań, autor pracy stworzył projekt i zaimplementował własne narzędzie, pozwalające na kooperacyjną edycję ontologii. Autor niniejszej pracy dokonał także wdrożenia systemu i przeprowadził jego testy.

Gdańsk, 10 grudnia 2008

Spis treści

Spis treści	i
1 Wstęp	3
1.1 Motywacje	3
1.2 Zakres pracy	5
2 Metody reprezentacji wiedzy i Sieć Semantyczna	7
2.1 Wprowadzenie	7
2.2 Wczesne metody reprezentacji wiedzy	7
2.2.1 Metody oparte o pojęcia ramek	10
2.2.2 Metody oparte o logiki opisowe	12
2.3 Sieć Semantyczna	16
2.3.1 Model opisu zasobów - RDF	17
2.3.2 Rodzina języków OWL	20
3 Przegląd dostępnych rozwiązań do edycji ontologii	29
3.1 Wprowadzenie	29
3.2 Przedstawienie wybranych edytorów	29
3.3 Opis metodologii oceny	31
3.4 Ocena wybranych rozwiązań	39
3.5 Podsumowanie	43
4 OCS - propozycja nowego systemu	45
4.1 Motywacje	45
4.2 Wymagania względem systemu OCS	46
4.2.1 Architektura logiczna systemu	48
4.3 Mechanizmy pracy zespołowej	50
4.3.1 Problemy pracy zespołowej	51

4.3.2	Model zarządzania wersjami	52
5	Analiza i projekt systemu	57
5.1	Wprowadzenie	57
5.2	Diagram przypadków użycia	57
5.2.1	Wprowadzenie	57
5.2.2	Aktorzy systemowi	58
5.2.3	Opis przypadków użycia	60
5.3	Projekt techniczny warstwy prezentacji	64
5.3.1	Podstawowe koncepcje rozwiązania	64
5.3.2	Diagram klas edytora ontologicznego	65
5.4	Struktura tworzonych plików	67
5.4.1	Plik konfiguracyjny edytora	67
5.4.2	Deskryptor projektu	69
6	Elementy implementacji	71
6.1	Wprowadzenie	71
6.2	Integracja z OWL API	71
6.3	Wdrażanie edytora ontologicznego	76
6.4	Silnik do wizualizacji grafów	79
6.5	Komponenty warstwy Web	82
6.6	Wynik implementacji	83
7	Przeprowadzone testy systemu	87
7.1	Wprowadzenie	87
7.2	Testy modułowe	87
7.3	Testy funkcjonalne	88
7.3.1	Specyfikacja i wyniki testów	89
7.4	Testy wydajnościowe	90
7.4.1	Specyfikacja i wyniki testów	90
8	Podsumowanie	97
	Bibliografia	99
A	Instrukcja obsługi systemu	103
A.1	Dostęp przez strony WWW	103
A.1.1	Struktura strony	103
A.1.2	Logowanie i rejestracja w systemie	104
A.1.3	Pobieranie edytora ontologicznego	105

A.1.4	Zarządzanie ontologiami	105
A.1.5	Zarządzanie użytkownikami	107
A.2	Edytor ontologiczny	107
A.2.1	Widok edytora	108
A.2.2	Pobieranie ontologii	109
A.2.3	Ładowanie i edycja ontologii	110
A.2.4	Przesyłanie zmian	111
A.2.5	Tworzenie nowej wersji	111
A.2.6	Eksport ontologii	111
B	Szczegółowy opis klas	113
B.1	Klasa <i>MainFrame</i>	113
B.2	Klasa <i>Logic</i>	113
B.3	Klasa <i>AbstractWizard</i>	114
B.4	Klasa <i>AbstractPage</i>	115
B.5	Klasa <i>GraphPanel</i>	115
B.6	Klasa <i>GraphDisplay</i>	115
B.7	Klasa <i>TreePanel</i>	116
B.8	Klasa <i>ComponentRepositoryPanel</i>	117
	Spis symboli i skrótów	119
	Spis rysunków	121
	Spis tabel	123

Podziękowania

Niniejsza praca powstała w ramach grantu KBN N516 035 31/3499 „Strategie i procedury tworzenia i negocjacji ontologii dziedzinowych”, realizowanego na Katedrze Architektury Systemów Komputerowych na Wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej. Powstały system nazwano OCS (ang. *Ontology Creation System*). Jego poszczególne moduły zostały wykonane w grupach, dzięki czemu stworzono kompletne środowisko, pozwalające na składowanie i pobieranie ontologii. Autor pracy odpowiedzialny był za stworzenie edytora ontologicznego oraz warstwy prezentacji systemu.

W tym momencie chciałbym podziękować drowi Pawłowi Czarnulowi, który nadzorował nasze prace, za zaangażowanie, jak również cenne uwagi. Chciałbym także podziękować osobom, które pracowały nad poszczególnymi modułami systemu: mgrowi Łukaszowi Budnikowi, Krzysztofowi Mazurkiewiczowi oraz Jackowi Mrozińskiemu. W szczególny sposób chciałbym podziękować mgrowi Tomkowi Boińskiemu, za „ducha optymizmu” oraz motywowanie naszego zespołu.

Dziękuję również rodzicom, za trud włożony w moje wychowanie oraz żonie - Ani, za wyrozumiałość oraz wsparcie w ciężkich chwilach.

Rozdział 1

Wstęp

1.1 Motywacje

W ostatnich latach wraz z gwałtownym rozwojem sieci Internet pojawiło się wiele problemów związanych z organizacją informacji. Mówi się, że Internet jest w stanie dostarczyć wszelkich danych, wystarczy tylko w umiejętny sposób ich szukać.

Pierwotna idea powstania sieci WWW (ang. *World Wide Web*) była związana ze sposobem przesyłania informacji poprzez globalną sieć. Na jej potrzeby zdefiniowano protokół HTTP (ang. *HyperText Transfer Protocol*), służący, w głównej mierze, do przesyłania dokumentów hipertekstowych, które opisane są w języku HTML (ang. *HyperText Markup Language*). Można więc powiedzieć, że Internet stał się pewną wspólną przestrzenią danych, dostępnych dla użytkowników. Początkowa idea budowy sieci WWW, nie zakładała jednak, że Internet rozrośnie się w tak szybkim tempie. Pomińto w ten sposób istotne aspekty mające na celu ułatwienie pozyskiwania danych z sieci WWW. Na bazie istniejącej Sieci powstało wiele narzędzi ułatwiających wyszukiwanie informacji, aczkolwiek okazało się, że problemem jest sama organizacja informacji w sieci WWW, w której dane pozbawione są treści zrozumiałej dla maszyn.

Dotychczasowy rozwój Internetu pokazał, że większość danych w nim umieszczanych jest zrozumiała tylko dla ludzi - komputery przetwarzają te dane jedynie w celach prezentacyjnych. Większość stron internetowych nie posiada żadnych metadanych, przez co ich automatyczne przetwarzanie staje się wręcz niemożliwe. Dlatego też powstała idea rozwoju obecnej sieci Web tak, aby informacje w niej zawarte były dobrze zdefiniowane i zawie-

rały treść (oprócz tej zrozumiałej tylko dla ludzi) także możliwą do automatycznego przetwarzania przez komputery. To podejście umożliwi lepszą współpracę ludzi z komputerami. U podstaw kolejnej generacji sieci WWW leży wizja Semantycznej Sieci Web, której autorem jest Tim Berners-Lee. Dzięki niej stanie się możliwa integracja wielu aplikacji oraz danych znajdujących się w Internecie. Bardzo istotnym aspektem tej wizji jest wiedza i sposób jej reprezentacji. W filozofii ontologia jest teorią badającą naturę bytów i ich strukturę. Nazwa ta została zaadaptowana do informatyki i oznacza dokument bądź plik, który w sposób formalny opisuje relacje pomiędzy terminami. Jest ona także odwzorowaniem fragmentu rzeczywistości.

W wyniku prac organizacji W3C (ang. *The World Wide Web Consortium*), powstało wiele standardów umożliwiających realizację wizji semantycznej sieci Web. U jej podstaw opierają się następujące elementy:

- **ontologie** - to formalna reprezentacja zbioru conceptów [10] w obrębie określonej dziedziny (biznes, nauka, medycyna), zawierająca także relacje między tymi conceptami,
- **bazy wiedzy** - to specjalne bazy danych wykorzystywane w procesach zarządzania wiedzą. Ich głównym zadaniem jest składowanie wiedzy, a także jej wydobywanie, w szczególności za pomocą mechanizmów wnioskujących,
- **środowiska agentowe** - to aplikacje działające w określonych środowiskach, zdolne do komunikowania się i monitorowania swojego otoczenia.

Nowoczesne podejście w procesie powstawania semantycznej sieci Web wymaga od ludzi zaangażowania w tworzeniu jej poszczególnych elementów. Powstało już wiele aplikacji prezentujących ciekawe wyniki przy wykorzystaniu ontologii, jednak prezentowane rozwiązania tworzone były na małą skalę. Semantyczna Sieć Web natomiast, pozwoli na przetestowanie ontologii na skalę światową. Wraz z powstaniem nowej gałęzi w informatyce i wyklarowaniem się standardów, przed inżynierem ontologii postawione zostały następujące zadania:

- **tworzenie ontologii** - definiowanie kolejnych ontologii, które są adaptowalne do konkretnej dziedziny,
- **scalanie ontologii** - użycie ontologii poprzez łączenie wielu ontologii z tej samej dziedziny w jedną ontologię ujednolicającą wszystkie łączone ontologie,

- **integracja ontologii** - wykorzystanie ontologii podczas tworzenia nowej ontologii w oparciu o ontologie już istniejące,
- **użycie ontologii** - stosowanie ontologii w konkretnych zastosowaniach, którymi mogą być: biznes, medycyna, nauka.

W celu realizacji tych zadań inżynierowi ontologii powinien być dostarczony zestaw narzędzi wspomagających jego pracę. Grupę narzędzi pozwalających na tworzenie i operowanie na ontologiach nazywamy edytorami ontologicznymi.

1.2 Zakres pracy

W ramach grantu, obejmującego stworzenie środowiska umożliwiającego składowanie i wykorzystanie ontologii, zdefiniowano założenia zakładające istnienie wspólnej platformy. Ukrywa ona przed użytkownikami sposób składowania ontologii i pozwala na skupienie się na praktycznym wykorzystaniu ontologii. Niniejsza praca magisterska poświęcona jest stworzeniu edytora ontologicznego, którego działanie opiera się na wykorzystaniu zdefiniowanej platformy. Praca nad całością systemu realizowana była w grupie 6 osobowej, której kierownikiem był dr inż. Paweł Czarnul. Autor w ramach pracy dyplomowej przygotował interfejs użytkownika systemu, co obejmowało zarówno stworzenie aplikacji samodzielnej (ang. *standalone application*), jak i stron internetowych. Praca ta dotyczyła także:

- **zaprojektowania systemu** - przedyskutowania i zaproponowania konkretnych rozwiązań w obrębie całej platformy ontologicznej, a także zaprojektowania edytora ontologicznego,
- **testowania** - jednostkowego, obejmującego poszczególne metody, jak i modułowego sprawdzającego działanie całego edytora,
- **integracji** - połączenia składowych modułów i udostępnienie aplikacji za pośrednictwem sieci WWW,
- **stworzenia testowej ontologii** - ukazującej działanie systemu.

W ramach pracy dyplomowej powstał prototyp systemu. Dostępny jest on pod adresem internetowym <http://ocs.kask.eti.pg.gda.pl>. Stworzona została także dokumentacja UML (ang. *Unified Modeling Language*) oraz Javadoc.

Powstały w ramach niniejszej pracy system ma szerokie zastosowanie w dziedzinie tworzenia ontologii. Szczególnie przydatny jest tam, gdzie w

grę wchodzi praca zespołowa dotycząca rozwijania ontologii. Może się także okazać przydatny, gdy istotny jest przyrostowy rozwój ontologii. Dzięki niemu można śledzić historię zmian i zawsze możliwy jest powrót do stabilnej wersji.

Rozdział 2

Metody reprezentacji wiedzy i Sieć Semantyczna

2.1 Wprowadzenie

Rozdział ten poświęcony jest zagadnieniom teoretycznym dotyczącym zarządzania wiedzą i sposobom jej reprezentacji. Opisane zostały w nim podstawowe metody reprezentacji ontologii, bazujące na pojęciach ramek oraz na formalizmach takich, jak logika opisowa. Ukazano również budowę Semantycznej Sieci Web, z położeniem nacisku na szczegółowe objaśnienie modelu opisu zasobów RDF (ang. *Resource Description Framework*) i schematu RDF - RDFS (ang. *Resource Description Framework Schema*). Rozdział ten wprowadza także czytelnika w bardziej ekspresywne języki - stosowane obecnie języki z rodziny OWL (ang. *Web Ontology Language*).

Rozdział został podzielony na następujące sekcje: 2.2 przedstawia sposoby reprezentacji wiedzy, w tym metody oparte o pojęcia ramek oraz logiki opisowe. Kolejna sekcja 2.3, przedstawia strukturę Semantycznej Sieci oraz szczegółowo opisuje języki RDF i OWL na bazie których, opiera się niniejsza praca.

2.2 Wczesne metody reprezentacji wiedzy

Od zarania dziejów ludzkość dążyła do zgromadzenia całej posiadanej wiedzy tak, aby stało się możliwe jej przekazanie kolejnym pokoleniom. Nasi przodkowie tworzyli w tym celu encyklopedie, słowniki, podręczniki i wiele

innych ksiąg. Większość z tej wiedzy została jednak spisana przy wykorzystaniu niestukturalnych metod. Uniemożliwia to wydajne przetwarzanie przez komputery.

W ostatnich dziesięcioleciach zaobserwowano intensywne prace nad strukturalnymi metodami zapisu wiedzy. W latach 1970-tych, kiedy dziedzina ta była bardzo popularna, dokonywano następującego podziału [2]:

- **formalizmy oparte o logikę matematyczną** - powstały z przekonania, że rachunek predykatów może być bezpośrednio użyty podczas wydobywania faktów o świecie,
- **inne** - opierały się na obserwacjach mechanizmów poznawczych u ludzi. Przykładem takich mechanizmów są struktury sieciowe lub metody oparte o reguły. Struktury sieciowe przypominały sposób odtwarzania wydarzeń analogiczny do pracy mózgu, gdzie fakty reprezentowane są za pomocą połączeń komórek nerwowych.

Wraz z powstaniem rachunku predykatów pierwszego rzędu, systemy oparte o logikę matematyczną okazały się bardziej obiecujące, gdyż mają silne podstawy matematyczne. Systemy takie zwykle wykorzystują jedynie podzbiór całego narzędzia jakim jest logika pierwszego rzędu. Wnioskowanie opiera się zaś na weryfikacji powstałych reguł logicznych.

W systemach nieopierających się o logikę matematyczną, mamy zwykle do czynienia ze specyficznymi strukturami, które wnoszą jakieś znaczenie. Wnioskowanie w takich systemach opiera się zwykle na operowaniu na tych strukturach, odkrywając połączenia między konkretnymi elementami. Wśród takich rozwiązań możemy wymienić: sieci semantyczne oraz systemy oparte o pojęcie ramek [10]. W obecnych czasach bardzo popularne stały się metody oparte o logiki opisowe (ang. *DL - Description Logic*), które korzystają z podzbioru rachunku predykatów pierwszego rzędu. Wynika to zapewne z inicjatywy Semantic Web, w ramach której powstał język OWL DL.

Należy jednak pamiętać, że od języka reprezentującego ontologię zależy efektywność jej wykorzystania. Dobry język opisu ontologii powinien być przede wszystkim jednoznaczny. Powinien także uwzględniać współpracę z narzędziem wnioskującym. Wnioskowanie jest istotne nie tylko ze względu na możliwości eksploracyjne ontologii (wnioskowania kolejnych faktów), lecz także w procesie oceny jakości ontologii [28]. Wnioskowanie może być wykorzystane w różnych momentach tworzenia ontologii np.: w czasie jej projektowania może posłużyć do określenia czy uzyskano pożądane relacje.

Innym popularnym wykorzystaniem wnioskowania jest określenie, czy dana klasa ma wystąpienia - określenie spełnialności danego konceptu.

W trakcie wdrażania ontologii bardzo istotnymi aspektami są także integracja i kooperacja ontologii z innymi ontologiami. Korzystając z mechanizmu wnioskującego możliwe jest np.: określenie, czy została zachowana spójność ontologii.

Poprzez wykorzystanie wnioskowania możliwe jest wykrycie części defektów w momencie tworzenia ontologii i usunięcie ich małym kosztem.

Wraz z powstaniem różnych sposobów reprezentacji wiedzy, powstają wymagania dotyczące języków ontologicznych. Do głównych z nich należą [28]:

- **dobrze zdefiniowana składnia** - jest wymagana ze względu na możliwości przetwarzania przez komputery. Języki powinny być oparte o dobrze zdefiniowaną bazę syntaktyczną, przez co możliwe jest wydajne przetwarzanie przez komputery,
- **dobrze zdefiniowana semantyka** - dotyczy głównie znaczenia pojęć występujących w danym języku. Przy użyciu języków opartych o dobrze zdefiniowaną semantykę, można przeprowadzić proces wnioskowania. Będzie ono dobrze funkcjonowało, gdy każde pojęcie będzie interpretowane tak samo przez każde narzędzie wnioskujące. Wymaganie to zwykle jest spełnione, gdy język reprezentacji ontologii opiera się o logikę matematyczną,
- **łatwość użycia** - to wymaganie dotyczy przede wszystkim użytkowników i określa, w jakim stopniu dany język reprezentacji ontologii jest przyjazny użytkownikowi. Wymaganie to było szczególnie istotne, gdy brakowało narzędzi wspomagających tworzenie i edycję ontologii. Należy tutaj nadmienić, że obecnie powstaje mnóstwo narzędzi, które ułatwiają tworzenie ontologii, ukrywając tym samym bezpośrednie wykorzystanie języka,
- **wystarczająca ekspresywność języka** - określa, w jakim stopniu możliwe jest wyrażenie wszystkich istotnych pojęć w stosunku do modelowanej dziedziny. Pozwala także określić, czy dany język jest adekwatny do zagadnienia, które chcemy opisać,
- **wydajne wsparcie dla wnioskowania** - stara się odpowiedzieć na pytanie jak bardzo wydajne jest wnioskowanie przy wykorzystaniu danego języka. Określa także, czy dany język jest rozstrzygalny, tzn.

odpowiada na pytanie czy istnieje efektywna metoda pozwalająca, w skończonej liczbie kroków, przypisać dowolne twierdzenie do tego języka.

Powyższe czynniki mogą posłużyć do wyboru odpowiedniego języka modelowania świata w postaci ontologii. W rzeczywistości wybór języka jest kompromisem pomiędzy jego ekspresywnością, a wydajnością wnioskowania.

2.2.1 Metody oparte o pojęcia ramek

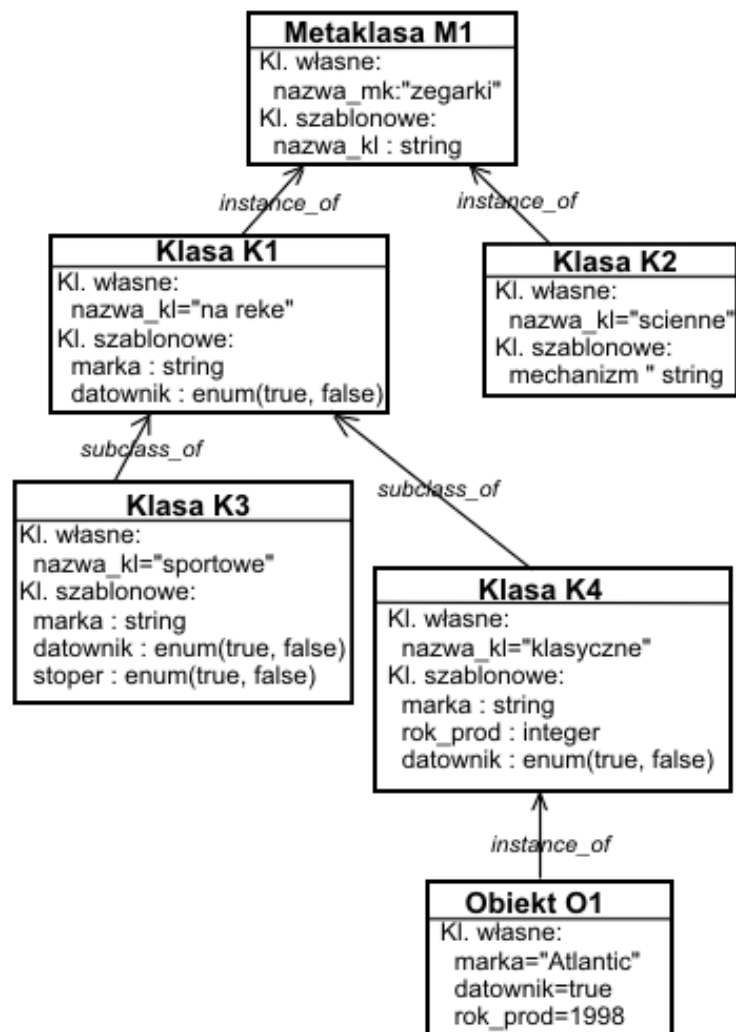
Termin „ontologia” określa dział filozofii dotyczący teorii bytu. Głównym kierunkiem działań filozofów zajmujących się ontologiami stały się badania struktury rzeczywistości. W informatyce, ontologia to formalny (strukturalny) sposób opisu pojęć występujących w modelowanej dziedzinie. Często ontologia określana jest także, jako fragment rzeczywistości odzwierciedlający powiązania pomiędzy jej elementami.

Jednymi z pierwszych sposobów reprezentacji wiedzy są ramki, opracowane przez Minsky’ego. Idea ta zakłada istnienie następujących elementów ontologii [10, 21]:

- **klasy** - to pojęcia występujące w danej dziedzinie,
- **klatki** (ang. *slots*) - opisują właściwości klasy. Istnieją dwa rodzaje klatek: własne (ang. *own*) i szablonowe (ang. *template*). Klatki własne opisują prywatne właściwości danej ramki, natomiast szablonowe służą do tworzenia nowych ramek poprzez wykorzystanie „szablonu”,
- **fasety** (ang. *facets*) - nakładają ograniczenia na klatki, np.: dziedzina, liczność, zakres,

Podjęcie Minsky’ego zakłada także istnienie metaklasy. Jest ona klasą, której wystąpienie ma swoje klatki szablonowe. Klasy można łączyć w hierarchie poprzez wykorzystanie relacji dziedziczenia. Dzięki temu klasy mogą „przekazywać” swoim potomkom klatki własne, jak również szablonowe. Obiektem staje się klasa, która nie może mieć wystąpień, a więc nieposiadająca klatek szablonowych. W celu zapewnienia kompletności opisu dostarczono aksjomatów, które dotyczą wszystkich elementów ontologii i są zapisane w języku logiki matematycznej.

Powyższe zależności zostały zobrazowane na rysunku 2.1. Poprzez analogię do przykładu ontologii „pojazdów” z [10], została na nim przedstawiona



Rysunek 2.1: Przykładowa ontologia wyrażona za pomocą ramek Minsky'ego.

ontologia *zegarków* wyrażona w języku ramek Minsky'ego. Na szczycie hierarchii znajduje się metaklasa M1, która posiada jedną klatkę własną „nazwa_mk” oraz jedną klatkę szablonową „nazwa_kl” („string” jest w rzeczywistości fasetem określającym dziedzinę klatki „nazwa_kl”). Metaklasa ma dwa wystąpienia - klasy „K1” i „K2”. Klatki szablonowe stały się klatkami własnymi z konkretnymi wartościami. Klasy K3 i K4 dziedziczą wszystkie klatki (zarówno własne jak i szablonowe) od klasy K1. Ramka O1 jest wystąpieniem klasy K4, tutaj obiektem, gdyż O1 nie posiada żadnych klatek szablonowych.

Systemy nie oparte o logikę matematyczną początkowo wydawały się

bardziej atrakcyjne od systemów bazujących na logice matematycznej z tego względu, że były bardziej nakierowane na postrzeganie świata przez ludzi. Niestety systemy te posiadały wiele wad, przez co skupiono się bardziej na systemach bazujących na rachunku predykatów pierwszego rzędu. Głównymi wadami reprezentacji ontologii przy wykorzystaniu ramek Minsky'ego są: niejednoznaczność i zbyt duży poziom abstrakcji. Spowodowane jest to brakiem ścisłej, formalnej definicji. Pokazano również, że pojęcie metaklasz prowadzi do nierozstrzygalności pewnych problemów wnioskowania [10]. Ramki Minsky'ego posłużyły jako podstawa do budowy formalizmu znanego jako F-Logic (ang. *Frames Logic*) [28].

2.2.2 Metody oparte o logiki opisowe

W związku z wadami metod reprezentacji wiedzy opierających się na pojęciu ramek i sieciach semantycznych, powstały próby formalizacji języka [2]. Początkowo zauważono, że można dokonać bardziej precyzyjnego opisu ontologii zdefiniowanej przy użyciu ramek, przy wykorzystaniu elementów logiki pierwszego rzędu. W wyniku intensywnych prac powstały tzw. systemy terminologiczne (ang. *terminological systems*), które możemy uznać za podstawę dzisiejszych logik opisowych. Początkowo nazwa „systemy terminologiczne” dotyczyła tego, że język był wykorzystywany do określenia terminologii (słownictwa) modelowanej dziedziny. Później nazwa przerodziła się w „języki konceptowe” (ang. *concept languages*). Jednak ostatecznie zaczęto używać nazwy „logiki opisowe”, która oznacza rodzinę języków reprezentacji wiedzy. Mogą być one wykorzystane do opisu wiedzy z określonej dziedziny w dobrze zrozumiałym i ustrukturalizowanym sposób [28, 2]. Logika opisowa ze względu na to, iż jest rozstrzygalnym podzbiorem rachunku predykatów pierwszego rzędu (ang. *FOL - First Order Logic*), nadaje się idealnie do automatycznego przetwarzania przez komputery.

Logika opisowa bazuje na następujących założeniach [2, 10]:

- ontologia opisuje pewne **uniwersum** - dziedzinę zainteresowań,
- elementami ontologii są **osobniki**, które definiują pojęcia występujące w danej dziedzinie,
- pojęcia występujące w ontologii powiązane są relacjami - **rolami**.

Zgodnie z powyższymi, ontologia wyrażona w języku logiki opisowej, składa się z następujących elementów: *TBox* (ang. *terminological box*) i *ABox* (ang. *assertional box*). *TBox* zawiera terminologię ontologii składającą się

Tabela 2.1: Przykładowe języki logik opisowych.

\mathcal{AL}	Podstawowy język, który pozwala na: <ul style="list-style-type: none"> • negację atomową, • przecięcie conceptów, • kwantyfikację egzystencjalną.
\mathcal{C}	Pozwala na pełną negację conceptu.
\mathcal{S}	Skrót dla \mathcal{AL} i \mathcal{C} , dla którego możliwe jest określenie ról przechodnich.
\mathcal{H}	Język, który pozwala na definiowanie hierarchii ról.
\mathcal{R}	Pozwala na definiowanie aksjomatów zawierania dla ról.
\mathcal{I}	Możliwość tworzenie ról przeciwnych.
\mathcal{N}	Możliwość definiowania ograniczeń liczebnościowych.
\mathcal{U}	Możliwość dokonania unii conceptów.

ze zbioru pojęć, ról, a także zbioru aksjomatów nakładających ograniczenia na concepty i role. Natomiast ABox zawiera wystąpienia pojęć i ról.

Istnieje wiele języków logiki opisowej, które mają elementy wspólne, np. [10]:

- **pojęcia atomowe** - między innymi concept uniwersalny \top , zwany także górnym, który reprezentuje naszą dziedzinę zainteresowań oraz concept pusty \perp , zwany również dolnym, który nie może mieć żadnych wystąpień,
- **role atomowe**,
- **konstruktory** - które umożliwiają tworzenie bardziej złożonych terminów.

Logiki opisowe, jak już poprzednio wspomniano, to rodzina języków o różnym stopniu ekspresywności. Wybór odpowiedniej, dla naszych potrzeb logiki opisowej, uzależniony jest od kompromisu pomiędzy ekspresywnością języka, a wydajnością wnioskowania.

Jednymi z bardziej popularnych języków zapisu ontologii, są języki oznaczane symbolami: \mathcal{ALC} , $\mathcal{SHOIN}^{(\mathcal{D})}$, $\mathcal{SROIQ}^{(\mathcal{D})}$, które mają zastosowanie m.in. w standardzie OWL DL, a także w aplikacji Protégé. Języki te wyposażone zostały w różne konstruktory, dzięki czemu możliwe jest zastosowanie różnych elementów języka, takich jak: negacja, przecięcie, unia, itp. W ta-

Tabela 2.2: Przykładowa ontologia prostego ekosystemu, bazująca na ontologii *rodziny* z [10].

TBox	ABox
$Roslina \sqsubseteq OrganizmZywy$	$Zwierze(kot)$
$Zwierze \sqsubseteq OrganizmZywy$	$Zwierze(mysz)$
$Zwierze \sqcap Roslina \equiv \perp$	$Roslina(pszenica)$
$Roslinozerca \equiv Zwierze \sqcap \exists zjada.Rosliny$	$zjada(kot, mysz)$
$Miesozerca \equiv Zwierze \sqcap \exists zjada.Roslinozercy$	$zjada(mysz, pszenica)$

beli 2.1 zostały przedstawione przykładowe języki zapisu ontologii wraz z wyjaśnieniami symboli języka [2].

W celu ilustracji działania języka \mathcal{ALC} przedstawiono przykładową ontologię (TBox oraz ABox) opisującą fragment ekosystemu. Ontologia ta bazuje na przykładzie ontologii „rodziny” przedstawionej w [10]. Zawiera ona następujące pojęcia: „OrganizmZywy”, „Zwierze”, „Roslina”, „Roslinozerca”, „Miesozerca” oraz rolę „zjada”. W tabeli 2.2 przedstawione zostały: terminologia oraz opis świata ontologii opisującej ekosystem (prosty łańcuch pokarmowy).

Terminologia tej ontologii zawiera pewne aksjomaty określające modelowaną rzeczywistość:

- zarówno Rośliny, jak i Zwierzęta są Organizmami Żywyymi,
- nie istnieją takie osobniki, które mogą być jednocześnie Roslina i Zwierciem,
- Roslinozerca, to takie Zwierze, które zjada Rosliny, natomiast Miesozerca to również zwierzę, które zjada Roslinozercow.

Pudełko asercjonalne zawiera natomiast rzeczywiste fakty o kocie, myszy i psie.

W wyniku obserwacji możemy dostrzec pewne dodatkowe fakty wynikające z zapisu powyższej ontologii: $Roslinozerca \sqsubseteq Zwierze$, $Miesozerca \sqsubseteq Zwierze$. W takim przypadku mówi się o niejawnym zawieraniu się pewnych pojęć. Jeśli natomiast do „pudełka” asercjonalnego dodalibyśmy następujące dwa fakty: $Zwierze(rosiczka)$, $Roslina(rosiczka)$, to rosiczka stałaby się osobnikiem bez konceptu (osobnikiem fałszywym). W takim przypadku baza wiedzy staje się sprzeczna [10].

Na podstawie powyższych rozważań możemy zauważyć, że definiując kilka konceptów możemy otrzymać taksonomię (uporządkowanie), która została zobrazowana w postaci hierarchii klas na rysunku 2.2.



Rysunek 2.2: Hierarchia dla ontologii ekosystemu.

Rozpatrując logiki opisowe, należy również omówić podstawowe problemy wnioskowania w ontologiach DL [10, 2]. Należą do nich:

- **zawieranie** (ang. *subsumption*) - stara się odpowiedzieć na pytanie, czy zbiór wystąpień konceptu C jest podzbiorem zbioru konceptów D ($C \sqsubseteq D$),
- **spełnialność** (ang. *satisfiability*) - określa, czy dany koncept ma wystąpienia ($C \equiv \perp$),
- **rozłączność** (ang. *disjointness*) - stara się odpowiedzieć na pytanie, czy zbiory wystąpień konceptów C i D są rozłączne ($C \sqcap D \equiv \perp$),
- **równoważność** (ang. *equivalence*) - odpowiada na pytanie, czy zbiory wystąpień konceptów C i D są równe ($C \equiv D$).

Jak można zauważyć, problemy wnioskowania nie są od siebie niezależne, wszystkie cztery można sprowadzić do problemu zawierania (subsumpcji):

- **spełnialność** C - czy $C \sqsubseteq \perp$?
- **równoważność** C i D - czy $C \sqsubseteq D$ i jednocześnie $D \sqsubseteq C$?
- **rozłączność** C i D - czy $C \sqsubseteq D \equiv \perp$ i jednocześnie $D \sqsubseteq C \equiv \perp$?

Na bazie powyższych pojęć wprowadza się aspekty jakościowe do baz wiedzy, które pozwalają na ocenę ontologii. Jednym z głównych pojęć jest **spójność ontologii** (ang. *consistency*), którą wyznacza się określając, czy wszystkie koncepty mają wystąpienia oraz czy w ontologii nie występują osobniki fałszywe (które nie są wystąpieniami żadnego konceptu).

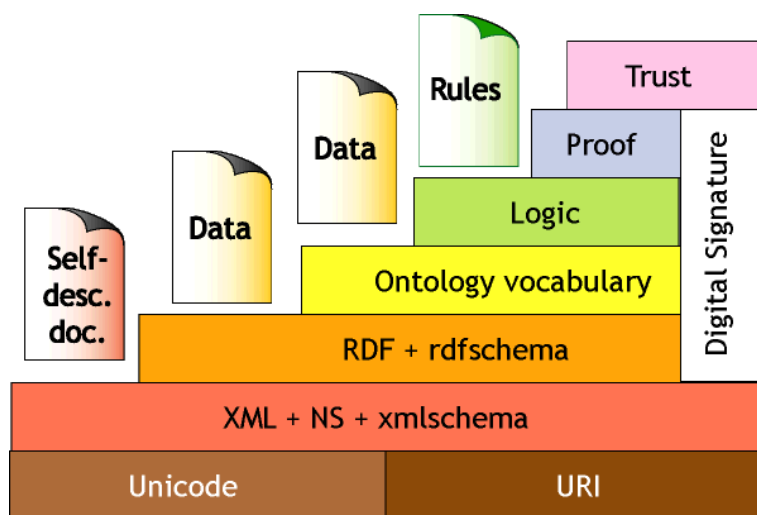
2.3 Sieć Semantyczna

W wyniku gwałtownego rozwoju Internetu powstała idea „porządkująca” zasoby WWW dotychczasowej Sieci. Podejście to zwykle się nazywało wizją Semantycznej Sieci Web. Idea ta miała na celu rozszerzenie istniejącej Sieci o mechanizmy umożliwiające automatyczne przetwarzanie zasobów. W ten sposób komputery mogłyby interpretować zawartość Sieci, która stałaby się dla nich „zrozumiała” (ang. *machine-readable*) [10].

Realizację powyższej idei ilustruje logiczna architektura Semantycznej Sieci Web (patrz rysunek 2.3). Powszechnie nazywana jest „tortem” Sieci Semantycznej (ang. *Semantic Web Cake*).

Idea ta zakłada istnienie danych, które oprócz właściwej treści (zrozumiałej dla ludzi), zawierają także treść możliwą do automatycznego przetwarzania przez komputery. Treść ta wyrażona jest przy wykorzystaniu języka XML, który służy jako baza składniowa dla nowych języków. Zgodnie z architekturą logiczną Semantycznej Sieci Web, model opisu zasobów (RDF) powinien posłużyć do budowy bardziej ekspresywnych języków ontologicznych. Ontologie wyrażone w kolejnych warstwach dadzą pewność, że wnioskowane fakty są prawdziwe i zgodne z rzeczywistością (warstwy *proof* i *trust*).

Całość pozwoli na zrealizowanie środowiska, w którym programy (agenty) w oparciu o ontologie, będą mogły przeprowadzić wnioskowanie. Użytkownicy Sieci będą zlecać swoje działania agentom, które wykorzystując



Rysunek 2.3: Struktura logiczna Semantycznej Sieci Web (diagram pochodzi ze strony www.semanticweb.org).

istniejące ontologie, będą realizowały złożone zadania. Przykładowym poleceniem wydanym agentowi może być: „Zarezerwuj mi wieczorny samolot z Londynu do Gdańska. Chciałbym również przenocować w pobliskim 4 gwiazdkowym hotelu w odległości nie większej jak 10km od lotniska.” Agent użytkownika będzie mógł się skontaktować z agentem odpowiedzialnym za rezerwacje biletów lotniczych, zlecić mu odnalezienie właściwego lotu i rezerwację biletu, po czym skorzysta z usług agenta odpowiedzialnego za rezerwacje noclegów i dobierze odpowiedni hotel. W ten sposób większość czynności będzie zrealizowana automatycznie.

Dzięki temu, że idea Semantycznej Sieci Web nie ingeruje w strukturę obecnej Sieci lecz ją rozszerza, możliwe będzie jej wdrożenie stopniowo, bez dokonywania gwałtownych zmian.

2.3.1 Model opisu zasobów - RDF

RDF to model opisu zasobów. Powstał jako rekomendacja organizacji W3C w celu opisu zasobów znajdujących się w sieci Internet [19]. RDF jest elementem, który stanowi podstawę wizji Semantycznej Sieci Web. Został opracowany jako wspólna platforma dająca możliwości stworzenia bardziej złożonych mechanizmów. Język ten został zaprojektowany na wysokim poziomie abstrakcji, dzięki temu możliwe jest modelowanie praktycznie wszystkich relacji. Jego składnia opiera się na prostej konstrukcji pozwalającej na opis zasobów w postaci trójek:

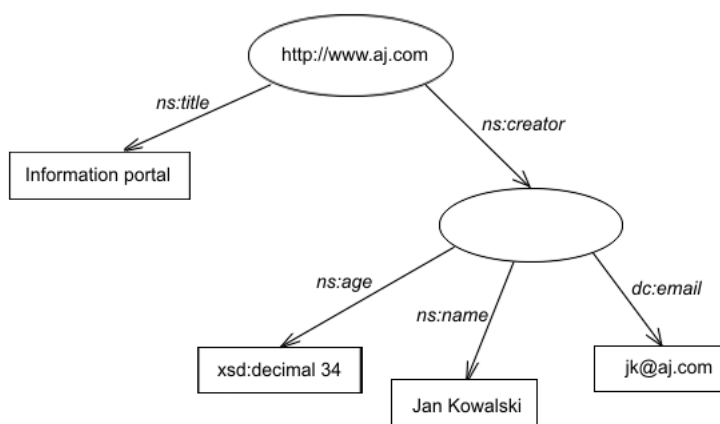
< podmiot, predykat, obiekt >

< podmiot, orzeczenie, dopełnienie >

< obiekt, właściwość, wartość >

Kolejną istotną cechą języka RDF jest to, że jest językiem zdecentralizowanym. W ten sposób bardzo dobrze nadaje się do środowiska sieci Internet, w której zasoby adresowane za pomocą URI nie muszą być konieczne osiągalne, a wręcz mogą nawet nie istnieć.

Każdy dokument RDF może być zaprezentowany w postaci grafu skierowanego, w którym zarówno węzły, jak i krawędzie, identyfikowane są za pośrednictwem URI. Przykładowy graf RDF został umieszczony na rysunku 2.4. Każdy taki graf może być interpretowany w następujący sposób:



Rysunek 2.4: Graf RDF przedstawiający opis strony WWW.

- eliptyczny węzeł z etykietą to zasób. Etykietą jest zawsze URI, które jednoznacznie identyfikuje zasób. Często URI są bardzo długie, przez co stają się nieporęczne w użyciu. Aby uniknąć tego problemu stosuje się przestrzenie nazw,
- eliptyczne węzły bez etykiety są węzłami anonimowymi (ang. *blank node*). Węzły anonimowe mają różne zastosowania [5]. Mogą być użyte do reprezentowania zasobów, które nie posiadają URI albo dla których URI nie jest znane. Ponadto mogą posłużyć do zdefiniowania pewnych struktur lub do zamodelowania n-krotnych relacji (podstawowa konstrukcja RDF - trójki, pozwalają jedynie na definiowanie relacji binarnych). Węzły anonimowe są bardzo podobne do kwantyfikacji egzystencjalnej z rachunku predykatów pierwszego rzędu, gdyż możemy powiedzieć, iż: „Istnieje choć jeden zasób o danych właściwościach.”,
- krawędzie w grafie oznaczają właściwość opisującą podmiot trójki. Na końcu krawędzi zakończonym strzałką, znajduje się węzeł określający wartość tej właściwości,
- kwadratowe węzły w grafie, oznaczają wartości literalne, który mogą, lecz nie muszą być konkretnego typu. W przypadku krawędzi *age*, wartość ta określona jest na podstawie XML Schema datatype, jako wartość decymalna wynosząca 34. Dla pozostałych krawędzi, w szcze-

gólności dla *name*, wartość literału nie ma typu i wynosi wprost „Jan Kowalski”.

Powyższy graf można zapisać w postaci następujących zdań:

1. `http://www.aj.com` ma tytuł (*ns:title*) „Information portal”.
2. `http://www.aj.com` został stworzony przez (*ns:creator*) twórcę.
3. Twórca (w rzeczywistości to węzeł anonimowy) ma 34 lata (*ns:age*).
4. Twórca nazywa się (*ns:name*) „Jan Kowalski”.
5. Twórca ma e-mail (*dc:email*) „jk@aj.com”.

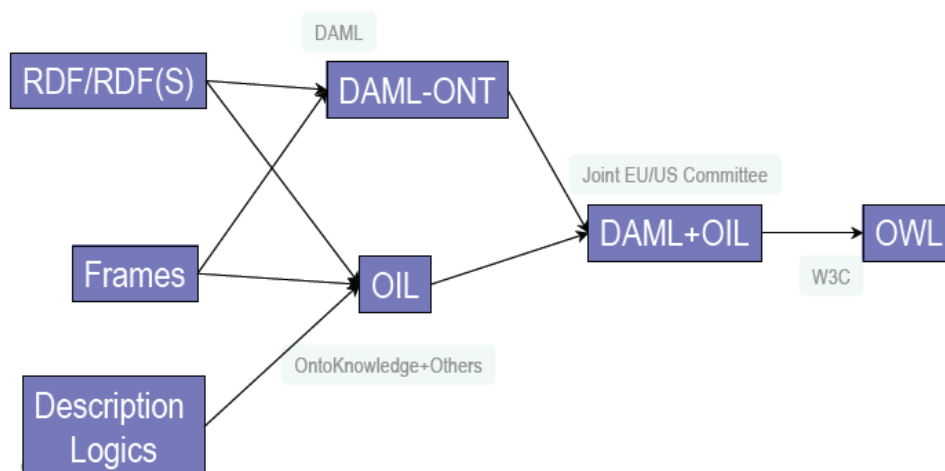
Poniższy listing przedstawia graf RDF, zaprezentowany przy użyciu oficjalnej składni zdefiniowanej w dokumencie [5].

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://dc#"
        xmlns:ns="http://ns#">

  <rdf:Resource rdf:about="http://www.aj.com"
                ns:title="Information portal">
    <ns:creator>
      <ns:name>Jan Kowalski</ns:name>
      <dc:email>jk@aj.com</dc:email>
      <ns:age
        rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        34
      </ns:age>
    </ns:creator>
  </rdf:Resource>

</rdf:RDF>
```

Standardem towarzyszącym RDF jest schemat RDF (RDFS). RDFS może być użyty do definiowania własnych klas węzłów oraz właściwości. Korzystając ze schematu RDF możemy stworzyć różne zależności pomiędzy klasami i właściwościami, przez co staje się możliwe wnioskowanie. Polega



Rysunek 2.5: Drzewo genealogiczne rodziny języków OWL.

ono na odkrywaniu kolejnych prawdziwych trójek na podstawie już dostarczonych trójek RDF. Jednak język ten ma niską ekspresywność, dlatego korzysta się z bardziej złożonych struktur: m.in.: z języków rodziny OWL.

2.3.2 Rodzina języków OWL

Ze względu na ograniczoną ekspresywność języka RDF i RDF Schema, zdecydowano się na wprowadzenie bardziej rozbudowanych języków reprezentacji wiedzy. W oparciu o podstawowe metody reprezentacji ontologii: ramki oraz logiki opisowe, stworzono język OIL (ang. *Ontology Inference Layer*). Równolegle, w ramach badań w USA zdefiniowany został język dla agentów - DAML (ang. *DARPA Agent Markup Language*). W oparciu o te dwa języki oraz model opisu zasobów powstała specyfikacja języka DAML+OIL (ang. *DARPA Agent Markup Language + Ontology Inference Layer*), która jest bezpośrednim przodkiem rodziny języków OWL. W skład OWL wchodzi trzy dialekty. Różnią się one ekspresywnością oraz możliwościami wnioskowania. Na rysunku 2.5, zostało przedstawione drzewo genealogiczne rodziny języków OWL.

Jak już prędzej wspomniano, rodzina OWL dostarcza języków o różnej sile ekspresji i różnych możliwościach wnioskowania. Dzięki temu użytkownik może zdecydować o doborze języka w stosunku do modelowanej dziedziny. W skład OWL wchodzi następujące dialekty [20]:

- **OWL Lite** - może być wykorzystany przez użytkowników, którzy

przede wszystkim chcą zbudować pewną hierarchię, jak również dodać proste ograniczenia. Niestety OWL Lite ma znaczne ograniczenia, np.: pozwala jedynie na ograniczenie liczności (ang. *cardinality*) do wartości 0 bądź 1. Język OWL Lite w większości operuje na klasach nazwanych, nie dopuszcza konstrukcji ze złożonymi opisami klas.

- **OWL DL** - posiada znacznie większe możliwości dotyczące siły ekspresji, jak i możliwości wnioskowania. OWL DL może być bezpośrednio mapowany na logikę opisową, a konkretnie na dialekt: $\mathcal{SHOIN}^{(D)}$. Dzięki temu wiadomo, że język ten jest rozstrzygalny i wszystkie obliczenia zakończą się w skończonym czasie [2].
- **OWL Full** - jest najbardziej rozbudowanym spośród pozostałych języków. Dostarcza największej siły ekspresji oraz w pełni wykorzystuje elastyczność RDF. Niestety język ten nie jest rozstrzygalny, a co za tym idzie nie mamy gwarancji, że obliczenia zakończą się w skończonym czasie. Wynika to z faktu, że OWL Full w odróżnieniu do DL, nie posiada wielu ograniczeń, w szczególności dotyczących właściwości przechodnich. Wymagane są one do podtrzymania rozstrzygalności języka [13].

Każdy z tych trzech języków jest rozszerzeniem swojego poprzednika. W związku z tym poniższe zdania są prawdziwe [20]:

- każda poprawna ontologia OWL Lite jest poprawną ontologią OWL DL,
- każda poprawna ontologia OWL DL jest poprawną ontologią OWL Full,
- każdy poprawny wniosek w OWL Lite jest poprawnym wnioskiem w OWL DL,
- każdy poprawny wniosek w OWL DL jest poprawnym wnioskiem w OWL Full.

Składnia języków OWL oparta jest o RDF i RDFS. Jednak nie wszystkie dialekty są w pełni kompatybilne z RDF. Największą kompatybilność zarówno syntaktyczną, jak i semantyczną, ma język OWL Full. Dla OWL DL i Lite zachodzi jedynie kompatybilność w „dół” (każdy poprawny dokument OWL DL, Lite jest także poprawnym dokumentem RDF, nie natomiast odwrotnie) [7].

Poniższy przykład dotyczy ontologii obrazującej proste zależności rodzinne. Możemy w niej wyróżnić następujące klasy: „Osoba”, „Mężczyzna”, „Kobieta”, „Rodzic”, „Ojciec”, „Matka”, „Dziecko”, „Syn”, „Córka”. Wszelkie relacje rodzinne można wyrazić w postaci następujących formuł, zapisanych w języku naturalnym jak również przy wykorzystaniu języka OWL DL:

Kobieta to Osoba, która nie jest Mężczyzną.

```
<!-- definicja klasy pojec kobieta, jako
podklasy osoba, dodatkowo klasa kobieta
jest rozlaczna z mezczyzna -->
<owl:Class rdf:ID="Kobieta">
  <rdfs:subClassOf rdf:resource="#Osoba" />
  <owl:disjointWith rdf:resource="#Mezczyzna" />
</owl:Class>
```

Podobnie: Mężczyzna jest Osobą, która nie jest Kobietą.

```
<owl:Class rdf:ID="Mezczyzna">
  <rdfs:subClassOf rdf:resource="#Osoba" />
  <owl:disjointWith rdf:resource="#Kobieta" />
</owl:Class>
```

Rodzic to taka Osoba, która ma przynajmniej jedno dziecko.

```
<!-- definicja klasy rodzic jako klasy rownowaznej
do klasy anonimowej, ktorej wystapienia podlegaja
ograniczeniu liczebnosciowemu (tutaj przynajmniej
1 dziecko) zwiazanemu z wlasciwoscia maDziecko -->
<owl:Class rdf:ID="Rodzic">
  <owl:equivalentClass>
    <!-- klasa rownowazna jest klasa anonimowa o ponizszej
    definicji -->
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
```

```

        <owl:onProperty rdf:resource="#maDziecko">
        <owl:minCardinality
            rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
            1
        </owl:minCardinality>
    </owl:Restriction>
    <owl:Class rdf:ID="Osoba"/>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>

<!-- definicja wlasciwosci maDziecko,
ktorej dziedzina i zakresem jest klasa osoba,
wlasciwosc maDziecko jest wlasciwością odwrotna
w stosunku do wlasciwosci maRodzica -->
<owl:ObjectProperty rdf:about="#maDziecko">
    <rdfs:domain rdf:resource="#Osoba"/>
    <rdfs:range rdf:resource="#Osoba"/>
    <owl:inverseOf rdf:resource="#maRodzica" />
</owl:ObjectProperty>

<!-- definicja wlasciwosci maRodzica -->
<owl:ObjectProperty rdf:about="#maRodzica">
    <!-- zdefiniowanie dziedziny i zakresu wlasciwosci,
        dla objectProperty dziedzina i zakres sa opisami klas -->
    <rdfs:domain rdf:resource="#Osoba"/>
    <rdfs:range rdf:resource="#Osoba"/>
    <owl:inverseOf rdf:resource="#maDziecko"/>
</owl:ObjectProperty>

```

Ojciec to Mezczyzna, będący Rodzicem.

```

<!-- definicja klasy ojciec jako klasy rownowaznej
do przeciecia (czesci wspolnej) klas rodzic
i mezczyzna -->
<owl:Class rdf:ID="Ojciec">
    <owl:equivalentClass>

```

```

<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Rodzic"/>
    <owl:Class rdf:about="#Mezyczna"/>
  </owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>

```

Matka to Kobieta, będąca Rodzicem.

```

<owl:Class rdf:ID="Matka">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Rodzic"/>
        <owl:Class rdf:about="#Kobieta"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

```

Dziecko to Osoba, która ma dwoje Rodziców.

```

<owl:Class rdf:about="#Dziecko">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="maRodzica"/>
          </owl:onProperty>
          <owl:cardinality
            rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
            2
          </owl:cardinality>
        </owl:Restriction>
        <owl:Class rdf:about="#Osoba"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

```

```

    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

```

Córka to Dziecko, będące Kobieta, ale nie będące Synem.

```

<owl:Class rdf:ID="Corka">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Dziecko"/>
        <owl:Class rdf:ID="Kobieta"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:disjointWith>
    <owl:Class rdf:ID="Syn"/>
  </owl:disjointWith>
</owl:Class>

```

Podobnie: Syn to Dziecko, będące Mężczyzną, ale nie będące Córką.

```

<owl:Class rdf:about="#Syn">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Mezczyzna"/>
        <owl:Class rdf:about="#Dziecko"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:disjointWith rdf:resource="#Corka"/>
</owl:Class>

```

Do tej definicji (pudełka „terminologicznego” - TBox) możemy dodać wystąpienia klas, wraz ze znanymi faktami, takimi jak: „Krzysztof ma dwoje rodziców: Annę i Jerzego.”, itp.

```
<!-- definicja poszczególnych bytów w ontologii
jako wystąpien klas -->
<Mezczyzna rdf:ID="Krzysztof">
  <maRodzica>
    <Kobieta rdf:ID="Anna" />
  </maRodzica>
  <maRodzica>
    <Mezczyzna rdf:ID="Jerzy">
      <maRodzica>
        <Mezczyzna rdf:ID="Jan" />
      </maRodzica>
      <maRodzica>
        <Mezczyzna rdf:ID="Maria" />
      </maRodzica>
    </Mezczyzna>
  </maRodzica>
</Mezczyzna>

<Mezczyzna rdf:ID="Jan" >
  <maDziecko rdf:ID="Zbigniew">
  </maDziecko>
  <maDziecko rdf:ID="Agnieszka">
  </maDziecko>
</Mezczyzna>

<Mezczyzna rdf:ID="Zbigniew">
</Mezczyzna>

<Mezczyzna rdf:ID="Marcin">
</Mezczyzna>

<Kobieta rdf:ID="Agnieszka">
</Kobieta>
```

Na podstawie powyższej ontologii możemy przeprowadzić wnioskowanie. Zadawane pytania mogą być różnorakie:

- **Podaj mi wszystkich synów Jana.** Mechanizm wnioskujący odpowie: Zbigniew, Jerzy.

- **Kto jest matką Krzysztofa.** Mechanizm wnioskujący odpowie: Anna.

Jednym z ciekawszych zapytań jest pytanie o rodziców Marcina. Mechanizm wnioskujący odpowie zbiorem pustym, gdyż tak naprawdę nie zna rodziców Marcina. Zakłada jednak, że taki osobnik istnieje. Baza wiedzy, w odróżnieniu do bazy danych, zakłada istnienie świata otwartego, stąd wie, że rodzice Marcina istnieją, lecz ich nie zna. Baza danych natomiast odpowie zbiorem pustym, gdyż informacje o rodzicach Marcina nie są w niej zapisane - opiera się na założeniu świata zamkniętego.

Rozdział 3

Przegląd dostępnych rozwiązań do edycji ontologii

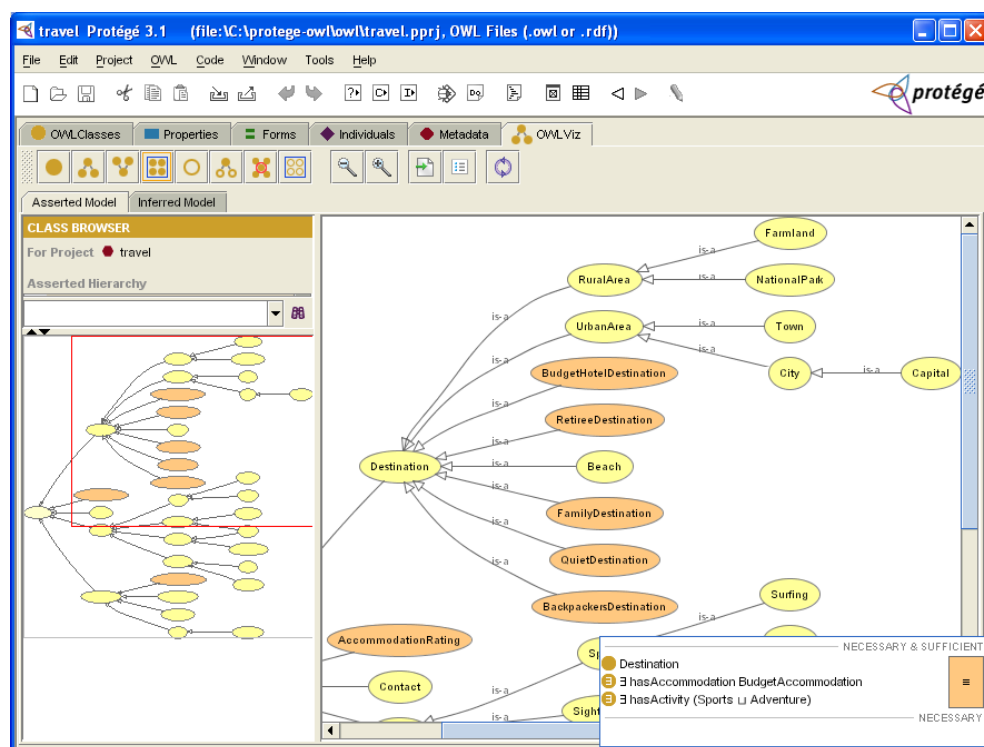
3.1 Wprowadzenie

W poniższym rozdziale przedstawiona została szczegółowa analiza rozwiązań w zakresie narzędzi do edycji ontologii. Zgodnie z przyjętą metodologią oceniono każdy edytor ontologiczny, po czym wskazano ten, który uzyskał najwyższą notę. Narzędzie to jest najbardziej dostosowane pod względem wymagań do tworzonego edytora ontologicznego dla systemu OCS.

W pierwszej części rozdziału nastąpi krótki opis rozwiązań, następnie będzie wyjaśniona metodologia oceny, po czym zostaną ocenione i porównane poszczególne edytory.

3.2 Przedstawienie wybranych edytorów

Obecnie na rynku dostępnych jest wiele rozwiązań pozwalających na tworzenie ontologii. Zdecydowana większość z nich rozwijana jest w środowiskach akademickich w oparciu o darmowe platformy technologiczne - w szczególności o język programowania Java. Ich cechą wspólną jest to, że starają się w mniejszym, bądź większym stopniu realizować lub wykorzystywać standardy organizacji W3C w zakresie zarządzania wiedzą. Większość z nich wykorzystuje graficzny interfejs użytkownika, przez co praca nad rozwijaniem ontologii staje się bardziej przyjazna użytkownikowi. Na rysunku 3.1 został przedstawiony zrzut ekranu z przykładowego edytora ontologii.



Rysunek 3.1: Zrzut ekranu z przykładowego edytora (tutaj Protégé).

Spośród dostępnych na rynku rozwiązań w dziedzinie edytorów ontologicznych najbardziej reprezentatywnymi przykładami są:

- **Protégé** - jest rozwiązaniem darmowym, z otwartym kodem, rozwijany jest na Uniwersytecie Stanforda. Jest silnie wspierany przez społeczność akademicką i ma zastosowania w dziedzinach począwszy od biomedycyny, skończywszy na biznesie. Jest bardzo dojrzałym rozwiązaniem zapewniającym możliwość rozwijania go za pomocą programów-wtyczek. Dostarcza mechanizmów wersjonowania i pracy grupowej. Jego kolejna wersja ma zostać przepisana na wspólny, obiektowy model OWL, przy wykorzystaniu biblioteki OWL API. Wspiera wiele formatów zapisu ontologii. Głównym atutem tego rozwiązania jest to, że jego rozwój jest silnie wspierany przez dużą społeczność, zarówno akademicką jak i rządową. Umożliwia tworzenie ontologii w formacie opartym o pojęcia ramek, a także w formacie zalecanym przez organizację W3C - OWL.

Strona domowa projektu: <http://protege.stanford.edu/>.

- **SWOOP** - to edytor, który był rozwijany w laboratoriach na Uniwer-

sytecie w Maryland. Nakierowany jest na tworzenie i edycję ontologii w formacie OWL. Jest rozwiązaniem mało dojrzałym. Początkowo rozwijany był w środowisku akademickim, obecnie jego kod został otwarty i udostępniony na stronie domowej projektu. Istnieje możliwość rozwijania tego narzędzia, niestety prace nad kolejnymi wydaniami nie są zbyt dobrze koordynowane. Ponieważ produkt cieszy się małym zainteresowaniem, stąd rzadko pojawiają się jego nowe wersje.
Strona domowa projektu: <http://code.google.com/p/swoop/>.

- **LinkFactory** - to rozwiązanie komercyjne, tworzone przez firmę L&C (Language&Computing). Pozwala na kooperacyjną pracę nad ontologiami. Umożliwia zapisanie ontologii w formacie OWL/RDF, przez co zapewnia zgodność ze standardem W3C. Za pośrednictwem tego narzędzia została rozwinięta, jedna z największych ontologii: LinkBase, będąca ontologią medyczną. Rozwiązanie opiera się na wersjonowaniu ontologii, dzięki czemu zachowany jest ślad zmian. Edytor ten może być zintegrowany z innymi systemami pozwalającymi na zarządzanie wiedzą. Jest rozwiązaniem najbardziej dojrzałym spośród wymienionych edytorów: został przetestowany w wielu dziedzinach: medycyna, prawo, obrona.

Strona domowa projektu: <http://www.landcglobal.com/>.

- **DOME** - rozwiązanie z otwartym kodem, które jest rozwijane w Instytucie DERI w Irlandii. Dostarcza pewnych mechanizmów rozszerzających popularne środowisko programistyczne Eclipse o wtyczki pozwalające edytować ontologie. Pozwala na wersjonowanie ontologii, dzięki czemu zostaje zachowana ciągłość informacji w zakresie tworzenia ontologii. Niestety zdecydowaną wadą jest to, że jakakolwiek praca wymaga znajomości struktury plików XML, do których zapisywana jest ontologia. Praktycznie nie istnieje możliwość graficznej edycji ontologii.

Strona domowa projektu: <http://dome.sourceforge.net/>.

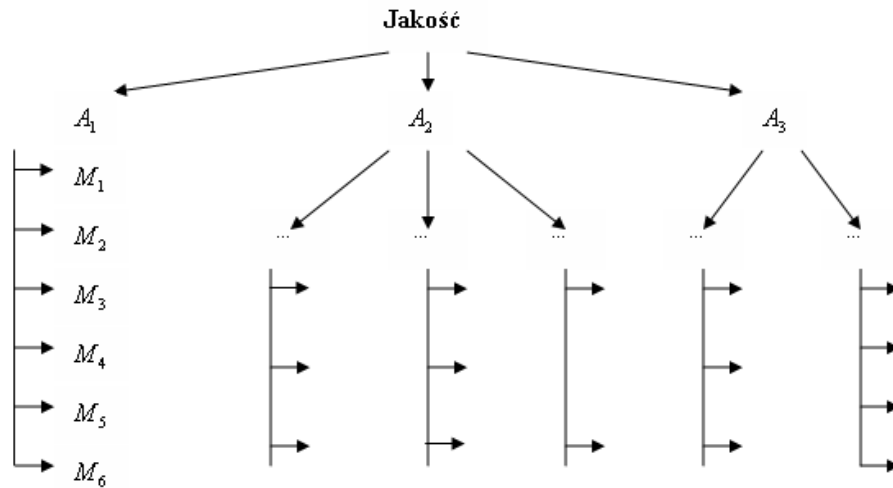
3.3 Opis metodologii oceny

Podczas analizy powyższych rozwiązań zdecydowano się na przyjęcie metodologii stosowanej w ramach oceny jakości [17, 18]. To podejście definiuje tzw. „drzewko jakości” na podstawie którego, wyznaczana jest ilościowa ocena jakości danego produktu. Metodologia ta wywodzi się z modelu pro-

ponowanego przez organizację ISO, który ma odzwierciedlenie w normie ISO/IEC 9126 [17]. Drzewko jakości zawiera dwa rodzaje węzłów:

- **atrybuty** - inaczej nazywane charakterystykami. Zwykle są bardzo ogólne, możliwe do przedstawienia w sposób opisowy. W rzeczywistości to wszystkie węzły w drzewie nie będące liśćmi. Definiują cechy jakościowe produktu, zwykle świadczą o wartości konkretnego produktu dla użytkownika. Węzły te posiadają wagi, które określają „istotność” atrybutu na danym poziomie drzewa jakości,
- **metryki** - to liście drzew jakości. Definiują cechy ilościowe, określające stopień „osiągnięcia” danego atrybutu. Węzły te obdarzone są odpowiednimi wagami, przez co możliwe jest wprowadzenie odpowiedniego poziomu „istotności” danej metryki w ocenie konkretnego atrybutu.

Istotnym aspektem, w momencie projektowania drzewa jakości, jest określenie odpowiednich atrybutów oraz metryk. Należy w tym celu wyznaczyć atrybuty złożone, po czym dokonać ich dekompozycji do momentu, gdy każdy atrybut da się opisać za pomocą metryk. Przykładowe drzewko jakości zostało przedstawione na rysunku 3.2



Rysunek 3.2: Przykładowe drzewo jakości.

W celu wyznaczenia wartości jakości danego produktu należy dokonać pomiarów wszystkich metryk i skorzystać ze wzoru na jakość: $J = \frac{\sum_i A_i \times w_i}{\sum_i w_i}$,

(gdzie A_i to kolejny atrybut będący bezpośrednim potomkiem węzła „jakość”, w_i określa wagę tego atrybutu, natomiast J to wyznaczana wartość jakości) [17]. Wyznaczenie wartości danego atrybutu może się odbywać w dwojaki sposób:

1. Dla atrybutów, których bezpośrednimi potomkami są inne atrybuty, należy skorzystać ze wzoru: $A = \frac{\sum_i A_i \times w_i}{\sum_i w_i}$, (gdzie A_i to kolejny atrybut będący bezpośrednim potomkiem atrybutu A , w_i określa wagę tego atrybutu, natomiast A to atrybut dla którego wyznaczana jest wartość).
2. Dla atrybutów, których bezpośrednimi potomkami są metryki, należy skorzystać ze wzoru: $A = \frac{\sum_i M_i \times w_i}{\sum_i w_i}$, (gdzie M_i to kolejny metryka będąca bezpośrednim potomkiem atrybutu A , w_i określa wagę tej metryki, natomiast A to atrybut, dla którego wyznaczana jest wartość).

Metoda ta opiera się na określeniu atrybutów, które bezpośrednio definiują jakość. Atrybuty te zwykle wskazują pewien aspekt, np.: kompletność, niezawodność. Dla powyższych atrybutów nie można dokonać bezpośredniego pomiaru, gdyż są to atrybuty złożone i zawierają wiele składowych oceny. Należy więc przeprowadzić dekompozycję tak, aby wyznaczyć atrybuty proste (niepodzielne), które są mierzalne. Atrybuty te to metryki. Metryki jednoznacznie określają sposób pomiaru danej cechy produktu [17, 18].

Dla powyższego modelu oceny jakości stworzono drzewo i przeprowadzono ocenę edytorów z punktu 3.2. Na poziomie pierwszym wyznaczono następujące atrybuty złożone:

- **funkcjonalność** - określa w jakim stopniu dany edytor ontologiczny jest dopasowany do wymagań wyspecyfikowanych względem edytora systemu OCS. **Waga:** 5,
- **wiarygodność** - określa stopień zaufania użytkownika do systemu, a także definiuje cechy edytora w zakresie zabezpieczenia edytora przed wyrządzeniem szkód na zewnątrz systemu oraz określa, w jakim stopniu system jest odporny na ingerencję niepożądanych systemów. **Waga:** 3,
- **elastyczność** - opisuje możliwości edytora w zakresie dopasowania się do zachodzących zmian. Określa także możliwości systemu pozwalające na uruchomienie go w różnych środowiskach, jak również opisuje w jakim stopniu użytkownik może wpłynąć na sposób funkcjonowania systemu. **Waga:** 2,

- **użyteczność** - określa, w jakim stopniu edytor ontologiczny udostępnia swoje funkcjonalności. Pozwala na przeprowadzenie oceny, czy dany edytor ontologiczny jest „przyjazny” użytkownikowi. **Waga:** 3.

Dla atrybutu złożonego **funkcjonalność**, wyznaczono następujące metryki:

Nazwa	Kompletność
Opis	Metryka określająca pokrycie funkcjonalności zdefiniowanych na liście pytań w polu „sposób pomiaru”.
Waga	3.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny odnośnie realizacji następujących funkcjonalności:</p> <ul style="list-style-type: none"> • Czy istnieje możliwość dodawania nowych elementów do ontologii? • Czy istnieje możliwość edycji istniejących elementów ontologii? • Czy istnieje możliwość usuwania istniejących elementów ontologii? • Czy istnieje możliwość pracy kooperacyjnej nad ontologiami? • Czy istnieje możliwość przeszukiwania ontologii względem występowania słów kluczowych? • Czy istnieje możliwość łączenia ontologii? • Czy istnieje możliwość oceny jakości ontologii, wedle ustalonych kryteriów? • Czy istnieje możliwość eksportu ontologii do różnych formatów? • Czy istnieje możliwość zaimportowania ontologii w różnych formatach do systemu? • Czy istnieje możliwość edycji z poziomu wizualizacji ontologii? • Czy edytor został wyposażony w jakikolwiek moduł wnioskujący?

Dla atrybutu złożonego **wiarygodność**, wyznaczono następujące metryki:

Nazwa	Ochrona
Opis	Określa stopień zabezpieczenia systemu przed dostępem niepożądanych użytkowników oraz stopień zabezpieczenia danych systemu przed zniszczeniem.
Waga	3.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny odnośnie zrealizowanych mechanizmów zabezpieczeń:</p> <ul style="list-style-type: none"> • Czy istnieje mechanizm uwierzytelniania i autoryzacji (0 - nie istnieją, 1 - istnieje uwierzytelnianie, 2 - istnieją oba mechanizmy)? • Czy istnieje mechanizm zabezpieczający przed utratą integralności danych?

Nazwa	Niezawodność
Opis	Określa liczbę upadków systemu w ściśle określonym czasie.
Waga	2.
Sposób pomiaru	<p>Odnótowanie liczby upadków systemu w czasie 1 - godzinnej pracy z narzędziem i przyznanie oceny zgodnie z następującymi kryteriami: powyżej 10 upadków - 0 pkt., 5-10 upadków - 1 pkt., 3-5 upadków - 2 pkt., 0-3 upadki - 3 pkt.</p>

Dla atrybutu złożonego **elastyczność**, wyznaczono następujące metryki i atrybuty:

Nazwa	Konfigurowalność
Opis	Określa stopień adaptowalności systemu do potrzeb użytkownika.
Waga	3.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny w zakresie adaptowalności systemu do potrzeb i wymogów użytkownika:</p> <ul style="list-style-type: none"> • Czy istnieje możliwość dostosowania interfejsu użytkownika? • Czy istnieje możliwość ustawienia różnych źródeł danych przechowujących ontologie? • Czy istnieje możliwość ustawienia internacjonalizacji?

Nazwa	Modyfikowalność
Opis	Określa możliwości w zakresie rozwoju systemu, dodawania do niego nowych funkcjonalności, edycji istniejących funkcjonalności, itp.
Waga	1.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny w zakresie możliwości modyfikowalności systemu:</p> <ul style="list-style-type: none"> • Czy kod źródłowy edytora jest otwarty? • Czy istnieje możliwość rozszerzania edytora na zasadzie programów- wtyczek?

Zdefiniowano również atrybut **przenośność**, określający możliwości systemu w zakresie przystosowania do różnych środowisk. **Waga:** 2.

Dla atrybutu **przenośność**, zdefiniowano następujące metryki:

Nazwa	Przenośność programowa
Opis	Określa możliwości systemu w zakresie przystosowania do różnych środowisk produkcyjnych.
Waga	3.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny w zakresie przenośności systemu:</p> <ul style="list-style-type: none"> • Czy istnieje możliwość uruchomienia systemu w systemach operacyjnych z rodziny Linux? • Czy istnieje możliwość uruchomienia systemu w systemach operacyjnych z rodziny MS Windows?

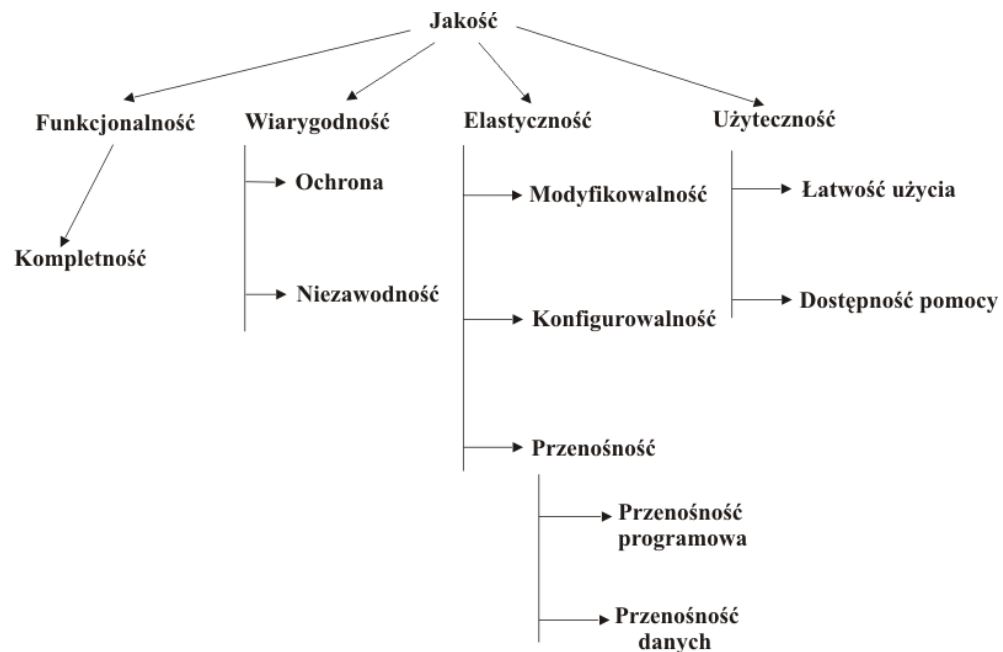
Nazwa	Przenośność danych
Opis	Określa możliwości systemu w zakresie przekształcania danych do różnych formatów.
Waga	4.
Sposób pomiaru	<ul style="list-style-type: none"> • Ocena: 2 punkty za możliwość zapisu do formatu zgodnego ze standardem OWL, za każdy dodatkowy format - ocena 1pkt. Łącznie nie więcej jak 5 punktów. • Ocena: 2 punkty za możliwość odczytu z formatu zgodnego ze standardem OWL, za każdy dodatkowy format - ocena 1pkt. Łącznie nie więcej jak 5 punktów.

Dla atrybutu złożonego **użyteczność**, wyznaczono następujące metryki:

Nazwa	Łatwość użycia
Opis	Określa stopień w jakim edytor ontologiczny jest „przyjazny” użytkownikowi.
Waga	3.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny w zakresie łatwości użycia danego edytora ontologii:</p> <ul style="list-style-type: none"> • Czy edytor wyposażony jest w graficzny interfejs użytkownika? • Czy edytor został podzielony na różne strefy reprezentujące dostęp do różnych funkcjonalności? • Czy istnieje repozytorium klas? • Czy istnieje repozytorium właściwości? • Czy istnieje repozytorium bytów? • Czy istnieje moduł wizualizacji ontologii?

Nazwa	Dostępność pomocy
Opis	Określa stopień dostępności pomocy w zakresie korzystania z systemu.
Waga	1.
Sposób pomiaru	<p>Odpowiedź na pytania i przyznanie oceny w zakresie dostępności pomocy do danego edytora ontologii:</p> <ul style="list-style-type: none"> • Czy wraz system został dostarczony podręcznik użytkownika? • Czy istnieje pomoc typu „samouczek”? • Czy istnieją możliwość skorzystania z pomocy kontekstowej? • Czy istnieje możliwość skorzystania z pomocy on-line?

Dla powyższej definicji drzewa jakości można wyznaczyć maksymalną ocenę. Korzystając ze wzoru (patrz punkt 3.3), po uwzględnieniu wszystkich wag i maksymalnych ocen otrzymujemy łączną ocenę: **482**.



Rysunek 3.3: Drzewo jakości zdefiniowane w celu oceny edytorów ontologicznych.

3.4 Ocena wybranych rozwiązań

W tym punkcie została zawarta ocena edytorów przedstawionych w punkcie 3.2, z wyłączeniem edytora komercyjnego LinKFactory rozwijanego przez firmę L&C (Language and Computing). Producent ten nie udostępnia żadnych wersji testowych. Podczas przeprowadzania oceny zastosowano metodologię „drzewka” jakości, opisaną w poprzednim punkcie. Dla każdego edytora został przeprowadzony test pod względem zgodności ze zdefiniowanymi atrybutami.

Wyniki analizy:

Protégé:

- Funkcjonalność

- **Kompletność:** Odpowiedź pozytywna na pytania: 1, 2, 3, 4, 5, 7, 8, 9, 11. Pozostałe odpowiedzi - negatywne. **Łącznie:** 9 pkt.

- **Wiarygodność**

- **Ochrona:** Przy zapisie danych w zewnętrznych repozytoriach wymagane jest zalogowanie: 2 pkt. Brak mechanizmów zapewniających integralność danych. **Łącznie:** 2 pkt.
- **Niezawodność:** W czasie 1 godzinnej pracy odnotowano 4 upadki. **Łącznie:** 2 pkt.

- **Elastyczność**

- **Modyfikowalność:** Kod edytora jest otwarty (1 pkt.), istnieje możliwość rozszerzenia edytora za pomocą programów-wtyczek (1 pkt.). **Łącznie:** 2 pkt.
- **Konfigurowalność:** Bogate możliwości ingerowania w interfejs użytkownika (1 pkt.). Możliwość ustawienia różnych źródeł ontologii (1pkt). Częstkowe wsparcie dla internacjonalizacji (0 pkt.). **Łącznie:** 2 pkt.
- **Przenośność:**
 - * **Przenośność programowa:** Możliwość pracy na systemach z zainstalowaną JVM, w szczególności na systemach z rodziny Linux (1 pkt.) i MS Windows (1 pkt.). **Łącznie:** 2 pkt.
 - * **Przenośność danych:** Eksport - 5 pkt., import - 4 pkt. **Łącznie:** 9 pkt.

- **Użyteczność**

- **Łatwość użycia:** Edytor został wyposażony w graficzny interfejs użytkownika (1 pkt.), przestrzeń robocza podzielona została na różne części, reprezentujące różne właściwości edytowanej ontologii (1 pkt.). Istnieją również wszystkie wymienione repozytoria (klas, właściwości, bytów) oraz moduł do wizualizacji ontologii. **Łącznie:** 6 pkt.
- **Dostępność pomocy:** Edytor udostępnia szerokie mechanizmy pomocy: pomoc kontekstowa (podpowiedzi typu tooltip - 1 pkt.), pomoc on-line (za pośrednictwem listy mailingowej - 1 pkt.), na stronie domowej produktu udostępnione zostały także podręczniki samouczki (1 pkt.) oraz dokumentacja produktu (1 pkt.). **Łącznie:** 4 pkt.

Podsumowanie:

Edytor **Protégé** jest produktem wartym zwrócenia uwagi. Jego silnymi stronami są metryki dotyczące: niezawodności i kompletności. Na pozytywną ocenę edytora mają także wpływ atrybuty: interfejs użytkownika i elastyczność.

Ostateczna ocena edytora: 415 (w przybliżeniu 86.10%).

SWOOP:

- **Funkcjonalność**

- **Kompletność:** Odpowiedzi pozytywne na pytania: 1, 2, 3, 8, 9. Pozostałe odpowiedzi - negatywne. **Łącznie:** 5 pkt.

- **Wiarygodność**

- **Ochrona:** Brak jakichkolwiek mechanizmów kontroli dostępu. **Łącznie:** 0 pkt.
- **Niezawodność:** W czasie 1 godzinnej pracy nie odnotowano żadnych upadków. **Łącznie:** 3 pkt.

- **Elastyczność**

- **Modyfikowalność:** Kod edytora jest otwarty (1 pkt.), nie ma możliwości rozszerzenia edytora za pomocą programów-wtyczek (0 pkt.). **Łącznie:** 1 pkt.
- **Konfigurowalność:** Istnieje możliwość ingerowania w interfejs użytkownika (1 pkt.). Możliwość wczytania ontologii z pliku bądź pobrania z predefiniowanych źródeł już istniejących ontologii (0 pkt.). Brak wsparcia dla internacjonalizacji (0 pkt.). **Łącznie:** 1 pkt.
- **Przenośność:**
 - * **Przenośność programowa:** Możliwość pracy na systemach z zainstalowaną JVM, w szczególności na systemach z rodziny Linux (1 pkt.) i MS Windows (1 pkt.). **Łącznie:** 2 pkt.
 - * **Przenośność danych:** Eksport - 5 pkt., import - 5 pkt. **Łącznie:** 10 pkt.

- **Użyteczność**

- **Łatwość użycia:** Edytor został wyposażony w graficzny interfejs użytkownika (1 pkt.), przestrzeń robocza podzielona została na różne części, reprezentujące różne właściwości edytowanej ontologii (1 pkt.). W edytorze występują wszystkie wymienione repozytoria (klas, właściwości, bytów), nie występuje jednak moduł wizualizacji ontologii. **Łącznie:** 5 pkt.
- **Dostępność pomocy:** Edytor pozbawiony jakiejkolwiek dokumentacji. Na stronie domowej projektu umieszczone są jedynie informacje odnośnie jego historii i osób zaangażowanych w jego tworzenie. **Łącznie:** 0 pkt.

Podsumowanie:

Edytor **SWOOP** jest produktem wartym zainteresowania, mimo mało dojrzałego procesu twórczego. Jego głównym atutem jest przejrzysty interfejs użytkownika oraz możliwość wersjonowania ontologii. Wypada znacznie gorzej w metryce „kompletność” w porównaniu z edytorem Protégé.

Ostateczna ocena edytora: 285 (w przybliżeniu 68.46%).

DOME:

- **Funkcjonalność**

- **Kompletność:** Odpowiedzi pozytywne na pytania: 1, 2, 3. Pozostałe odpowiedzi - negatywne. **Łącznie:** 3 pkt.

- **Wiarygodność**

- **Ochrona:** Ponieważ projekt jest ukierunkowany na zespołowe tworzenie ontologii, w szczególności przy wykorzystaniu repozytorium CVS, na projekt zostały narzucone wymagania odnośnie ochrony związane z samym środowiskiem CVS. **Łącznie:** 3 pkt.
- **Niezawodność:** W czasie 1 godzinnej pracy nie odnotowano żadnych upadków. **Łącznie:** 3 pkt.

- **Elastyczność**

- **Modyfikowalność:** Kod edytora jest otwarty, istnieje możliwość rozszerzania edytora na zasadzie programów wtyczek, przy wykorzystaniu mechanizmów ze środowiska Eclipse. **Łącznie:** 2 pkt.

- **Konfigurowalność:** Brak jakichkolwiek mechanizmów pozwalających na dostosowanie interfejsu użytkownika do własnych potrzeb (0 pkt.). Możliwość wczytania ontologii z pliku bądź pobrania z repozytorium CVS (1 pkt.). Brak wsparcia dla internacjonalizacji (0 pkt.). **Łącznie:** 1 pkt.
- **Przenośność:**
 - * **Przenośność programowa:** Możliwość pracy na systemach z zainstalowaną JVM, w szczególności na systemach z rodziny Linux (1 pkt.) i MS Windows (1 pkt.). **Łącznie:** 2 pkt.
 - * **Przenośność danych:** Eksport - 1 pkt., import - 1 pkt. **Łącznie:** 2 pkt.
- **Użyteczność**
 - **Łatwość użycia:** Edytor został wyposażony w graficzny interfejs użytkownika (1 pkt.), przestrzeń robocza nie jest podzielona na różne części, reprezentujące, przez co praca na dużych ontologiach staje się trudna (0 pkt.). Brak modułu wizualizacji ontologii. **Łącznie:** 1 pkt.
 - **Dostępność pomocy:** Edytor pozbawiony jakiegokolwiek dokumentacji. Na stronie domowej umieszczone zostały informacje przydatne dla programistów - dokumentacja JavaDoc (1 pkt.). **Łącznie:** 1 pkt.

Podsumowanie:

Edytor **DOME** jest narzędziem, które nie pozwala na wydajne tworzenie ontologii, gdyż w interfejsie użytkownika nie ma jakiegokolwiek grupowania. Dodatkowo nie wspiera języków zdefiniowanych w standardzie OWL. Głównym jego atutem jest zorientowanie na pracę grupową. Zdecydowanie jest najsłabszym edytorem wśród porównywanych.

Ostateczna ocena edytora: 168 (w przybliżeniu 34.85%).

3.5 Podsumowanie

Zaprojektowane drzewko jakości promuje najbardziej reprezentatywne cechy edytora ontologicznego dla systemu OCS. Przeprowadzona analiza ukazuje stopień dopasowania poszczególnych edytorów do drzewka jakości. Należy tutaj dodać, że ta metodologia nie uwzględnia elastyczności architektury systemu OCS, gdyż nie jest ona tematem niniejszej pracy.

Najlepiej spośród wszystkich narzędzi umożliwiających edycję ontologii wypadł **Protégé**. Uzyskał notę 415 punktów na 482 możliwych. Znacznie gorzej wypadł edytor **SWOOP**, który uzyskał ocenę 285 punktów, oraz **DOMÉ** - tylko 168 punktów. W przypadku dwóch pierwszych edytorów istnieje możliwość ich rozszerzenia, co może się okazać bardzo przydatne, biorąc pod uwagę ich adaptację do architektury systemu OCS.

Rozdział 4

OCS - propozycja nowego systemu

4.1 Motywacje

Odkąd języki Semantycznego Internetu (RDF, OWL) powstały i są powszechnie wykorzystywane, zaistniała potrzeba dostarczania kolejnych narzędzi umożliwiających ich wykorzystanie. Narzędzia te mogą mieć różnorakie zastosowanie: edycja i rozwój ontologii, wnioskowanie, składowanie ontologii, itp. Autor niniejszej pracy zaimplementował część związaną z warstwą prezentacji dla systemu, który nosi nazwę OCS - system składowania i pozyskiwania ontologii. Główną częścią warstwy prezentacji stał się edytor ontologiczny, który w porównaniu z rozwiązaniami przedstawionymi w rozdziale 3, wyróżnia się następującymi cechami:

- **kooperacyjna edycja ontologii** - edytor pozwala na rozwój ontologii w zespole,
- **koordynacja prac** - nowe wersje ontologii publikowane są przez uprzywilejowanych użytkowników,
- **integracja z systemem OCS** - edytor został zintegrowany z systemem składowania i pozyskiwania ontologii, przez co stanowi nierozłączną całość,
- **przejrzysty interfejs użytkownika** - główną część edytora stanowi panel odpowiedzialny za graficzną wizualizację elementów ontologii, natomiast ich edycja odbywa się w repozytorium komponentów,

- **wielojęzyczność** - edytor jest platformą wielojęzyczną. W pierwszej wersji ma dwa tłumaczenia: na język polski i angielski,
- **łatwa instalacja przez sieć WWW** - edytor został opublikowany w sieci WWW przy wykorzystaniu technologii aplikacji klienckiej, w pełni automatyzującej proces instalacji i administracji. Aby go pobrać, wystarczy kliknąć na hiperłącze. Nie trzeba się martwić o aktualizacje - wszystko odbywa się automatycznie.

4.2 Wymagania względem systemu OCS

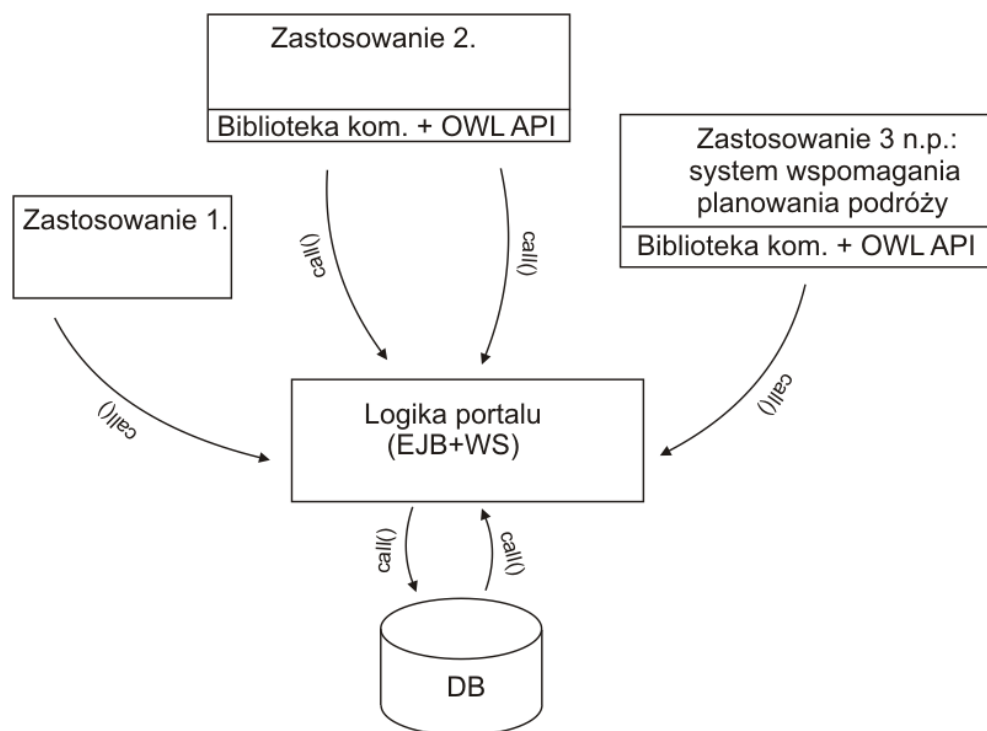
W ramach spotkań katedralnych, opracowano zestaw założeń dotyczących systemu OCS oraz wskazano jego cele biznesowe. Określono również interfejs programistyczny (ang. *API - Application Programming Interface*), pozwalający na wykonanie podstawowych funkcjonalności systemu. Główne wymagania stawiane względem systemu OCS:

- tworzony system powinien umożliwiać praktyczne wykorzystanie budowanych ontologii w dziedzinach: biznes, bezpieczeństwo, medycyna,
- jednym z komponentów systemu powinno być środowisko do tworzenia i analizy ontologii - edytor ontologiczny,
- powinny istnieć mechanizmy usprawniające komunikację pomiędzy specjalistami tworzącymi ontologie,
- poszczególne komponenty systemu powinny korzystać ze **wspólnej platformy**, przeznaczonej do składowania i pozyskiwania ontologii, tak aby użytkownik systemu mógł skupić się nad wykorzystaniem ontologii, nie zaś nad uzyskaniem do niej dostępu,
- powinna istnieć możliwość podłączenia do systemu różnych „**zastosowań**”. Będą one korzystały ze wspólnego interfejsu programistycznego. Zastosowaniami mogą być wszelkiego rodzaju systemy, które w oparciu o ontologie, wspomagają użytkownika w planowaniu, np.: system wspomagania planowania podróży, który ułatwia wybór trasy, pozwala na rezerwację miejsc w hotelach oraz organizację czasu wolnego,
- system ma umożliwiać składowanie ontologii w **różnych formatach**, dlatego format zapisu ontologii powinien być elastyczny (przyjęto model trójek RDF),

- system, po stronie serwera aplikacji, **nie powinien wykonywać złożonego przetwarzania**,
- dostęp do ontologii, oprócz reprezentacji trójkowej, powinien być zrealizowany w bardziej „przyjazny” sposób. Zdecydowano się na wykorzystanie gotowej implementacji modelu OWL - OWL API. Pozwala ona na bezpośrednie mapowanie trójek RDF na model obiektowy języka OWL,
- system powinien być zrealizowany w oparciu o **darmowe rozwiązania**, bazujące na języku *Java*.

Należy tutaj dodać, że zaplanowano iż praca nad systemem będzie realizowana od podstaw, wspólnymi siłami Katedry, cyklem wydaniowym. Pierwsze wydanie systemu będzie *de facto* prototypem. Kolejne wydania będą dostosowywały system do potrzeb jego użytkowników.

Najlepszym odwzorowaniem dla powyższych wymagań, pokazującym sposoby wykorzystania systemu, jest rysunek 4.1. Przedstawiono na nim



Rysunek 4.1: Sposoby wykorzystania systemu OCS.

wysokopoziomą wizję systemu, w ramach której zdefiniowano podstawowe komponenty systemu:

- **baza danych** - jest składowiskiem ontologii. W bazie danych przechowywane są wszystkie ontologie wraz z informacjami o wersjach, użytkownikach systemu, itp.,
- **warstwa logiki portalu** - udostępnia wszelkie metody biznesowe za pośrednictwem komponentów EJB (ang. *Enterprise Java Beans*) oraz wołań usług sieciowych (ang. *Web Services*). Metody te pozwalają na przezroczysty, z punktu widzenia użytkownika, dostęp do ontologii,
- **biblioteka komunikacyjna** - „opakowuje” wywołania metod biznesowych zdefiniowanych w warstwie logiki portalu, dzięki czemu dostęp do ontologii staje się jeszcze bardziej przyjazny użytkownikowi systemu. Biblioteka ta wykorzystuje również OWL API, w celu obiektowej reprezentacji ontologii,
- **warstwa zastosowań** - wykorzystuje metody zdefiniowane w warstwie logiki portalu, bądź w bibliotece komunikacyjnej. Warstwa ta reprezentuje wszelkie możliwe sposoby wykorzystania systemu, którymi mogą być: wszelkie systemy stowarzyszone, korzystające z ontologii przechowywanych w systemie OCS.

4.2.1 Architektura logiczna systemu

W wyniku dalszej analizy oraz w oparciu o specyfikację wymagań, uszczegółowiono wizję systemową. Edytor ontologiczny, omawiany w ramach tej pracy, stał się *de facto* częścią systemu OCS. Narzędzie to miało być wykorzystywane do tworzenia i rozwijania ontologii, składowanych w systemie OCS. Bazując na powyższych założeniach oraz w oparciu o ustalony podział prac, zdecydowano się na przyjęcie konkretnego rozwiązania opierającego się na architekturze aplikacji wielowarstwowych. W tym momencie przedyskutowano również dostępne darmowe rozwiązania mogące ułatwić implementację systemu. W ramach spotkań grupy odpowiedzialnej za stworzenie systemu zidentyfikowano następujące warstwy:

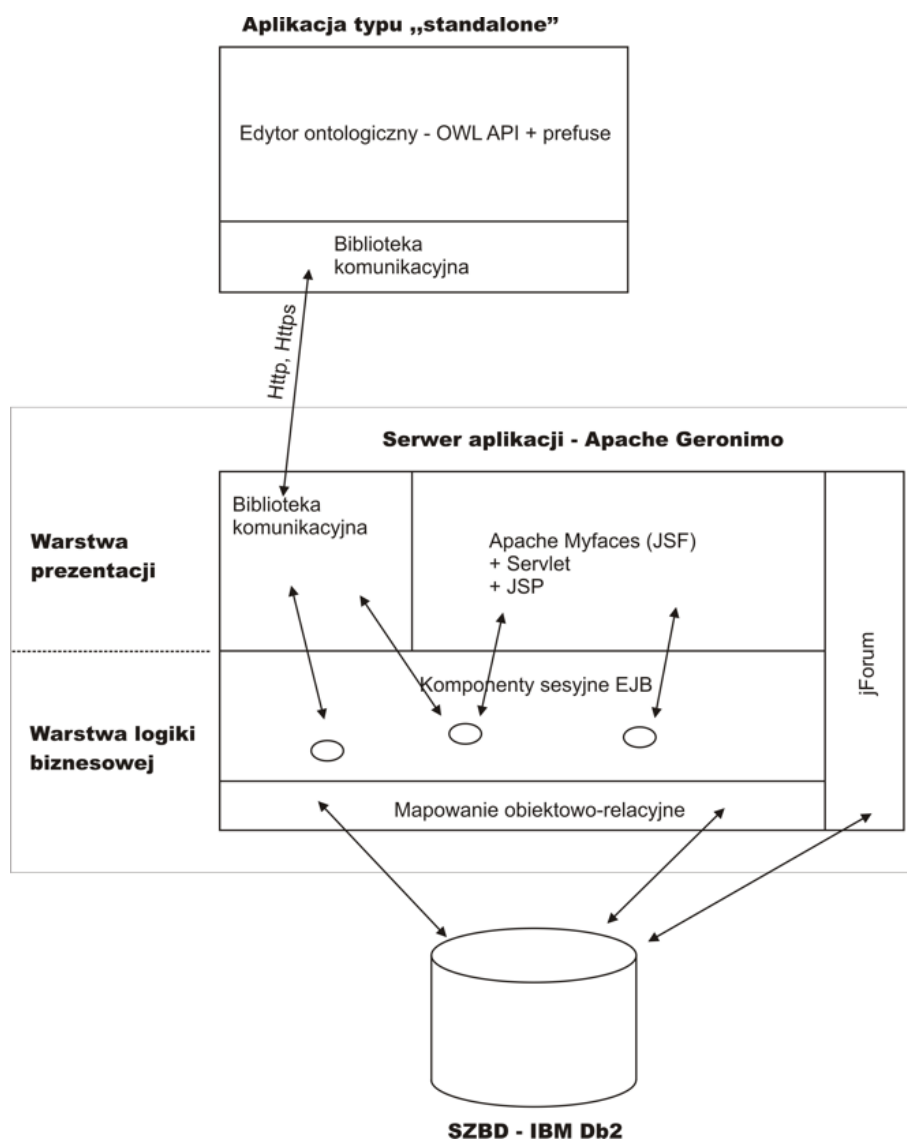
- **warstwa prezentacji** - stanowią ją *serwlety* i strony *jsp*. W związku z większym uporządkowaniem kodu źródłowego zdecydowano się na wykorzystanie specyfikacji JSF (ang. *Java Server Faces*). Specyfikacja ta pozwala na tworzenie aplikacji web'owych, przy wykorzystaniu wzorca projektowego MVC (ang. *Model View Controller*). Wzorzec ten pozwala na odseparowanie logiki dostępu do danych (model) od prezen-

tacji (widoku). Jako rozwiązanie projektowe przyjęto implementację specyfikacji JSF - Apache MyFaces.

- **warstwa logiki biznesowej** - stanowią ją komponenty serwera aplikacji - sesyjne EJB. W warstwie tej ma zostać zrealizowana cała logika biznesowa. Komponenty EJB mają udostępniać swoje funkcjonalności zarówno za pośrednictwem rejestru JNDI (ang. *Java Naming and Directory Interface*), jak również jako usługi sieciowe.
- **warstwa dostępu do danych** - opiera się na „silniku” mapowania obiektowo-relacyjnego (ang. *Object-Relational mapping*). Dzięki temu wszystkie operacje wykonywane na danych, są niezależne od doboru serwera baz danych.
- **model danych** - składa się z odpowiednich tabel relacyjnej bazy danych, korzystającej z języka zapytań SQL. Zdecydowano, iż baza danych będzie rezydowała na serwerze DB2.

Na rysunku 4.2 przedstawiona została architektura logiczna systemu OCS. Najciekawszym elementem systemu jest biblioteka komunikacyjna. „Opakowuje” ona metody biznesowe zdefiniowane w warstwie logiki portalu tak, że stają się one dostępne w rozproszonym środowisku internetowym. Użytkownik korzystając z tej biblioteki jest w stanie wykonywać wszelkie operacje na ontologii w postaci wywołań lokalnych procedur. Jednak, w rzeczywistości, metody te wykonują zdalnewołanie metod logiki portalu znajdujących się na serwerze systemu OCS. Komunikacja pomiędzy biblioteką a serwerem aplikacji odbywa się w oparciu o powszechnie znane protokoły HTTP i HTTPS (ang. *HyperText Transfer Protocol Secure*). Niewątpliwymi atutami powyższego rozwiązania są:

- komunikacja nieblokowana przez zapory typu *firewall*,
- pełna integracja z serwerem aplikacji,
- wsparcie dla procesu uwierzytelniania, poprzez wykorzystanie wbudowanych w kontener serwletów mechanizmów uwierzytelniania. Jednym z nich jest chronienie dostępu do zasobów poprzez użycie metody HTTP Basic. Polega ona na przesłaniu przez serwer odpowiedzi z prośbą o przesłanie odpowiednich danych do uwierzytelnienia, w momencie próby dostępu do zasobów chronionych. Dane te przesyłane są w nagłówku żądania HTTP (pole „Authorization”) i tworzona jest sesja użytkownika,



Rysunek 4.2: Architektura logiczna systemu OCS.

- w przypadku HTTPS, komunikacja odbywa się bezpiecznym, szyfrowanym kanałem.

4.3 Mechanizmy pracy zespołowej

W celu zaprezentowania szerokiego *spectrum* problemów związanych z pracą zespołową, scharakteryzowano podstawowe wymagania dotyczące pracy w grupie oraz wybrane rozwiązania wykorzystane w systemie OCS.

4.3.1 Problemy pracy zespołowej

Doświadczenia wielu pokoleń pokazały, że praca zespołowa jest bardzo przydatna zwłaszcza, gdy w grę wchodzi praca nad ogromnym przedsięwzięciem. Obecnie możemy zauważyć, że większość projektów informatycznych dotyczy coraz większych problemów, którym jednostka nie jest w stanie poddać. Podobnie, problem ten dotyczy również systemów związanych z gromadzeniem wiedzy. Często zdarza się, że modelowana wiedza - ontologia, dotyczy bardzo szerokiej dziedziny. Pojedyncza osoba nie jest w stanie wyrazić wszystkich istotnych pojęć, gdyż po prostu ich nie zna. W związku z tym idea pracy zespołowej, wykorzystana w systemach gromadzenia wiedzy, może przyczynić się do bardziej efektywnego tworzenia ontologii. Poza tym, w zespole często mamy do czynienia z efektem synergicznym, w którym grupa wnosi więcej niż sumarycznie każdy z członków osobno [14]. Jednak w parze z wieloma walorami pracy zespołowej, powstaje również wiele problemów, które mogą przyczynić się do obniżenia efektywności prac. Do głównych aspektów zapewniających większą efektywność pracy grupowej należą:

- **komunikacja** - powinna dostarczać wszelkich mechanizmów pozwalających na nawiązanie komunikacji pomiędzy członkami zespołu. Mogą do nich należeć mechanizmy typu: e-mail, fora internetowe, czaty, konferencje głosowe, wideokonferencje. Komunikacja w zespole jest czynnikiem warunkującym sukces przedsięwzięcia,
- **koordynacja** - prace muszą być koordynowane. Innymi słowy, w zespole powinna zostać wyróżniona osoba, która przyjmuje rolę lidera i czuwa nad wszelkimi poczynaniami innych osób. Bez jakiegokolwiek koordynacji praca w zespole staje się chaotyczna, przez co znacznie zmniejsza się jej efektywność,
- **kooperacja** - czyli współpraca wielu członków nad wspólną częścią, bądź nad różnymi fragmentami systemu. Jest w rzeczywistości najtrudniejsza do realizacji, gdyż wymaga dostarczenia mechanizmów synchronizacji, jak również blokowania w celu wykluczenia wzajemnie nakładających się prac. W celu zapewnienia dobrej współpracy wymagane jest także, takie zaprojektowanie systemu, aby dla jego użytkowników zniesione zostały wszelkie bariery geograficzne, językowe, czasowe, itp.

Sformułowane powyżej problemy noszą nazwę zasady 3xK. Zasada ta jest powszechnie uważana za podstawę efektywnej pracy zespołowej i od jej wypełnienia, w mniejszym bądź większym stopniu, zależy skuteczność grupy.

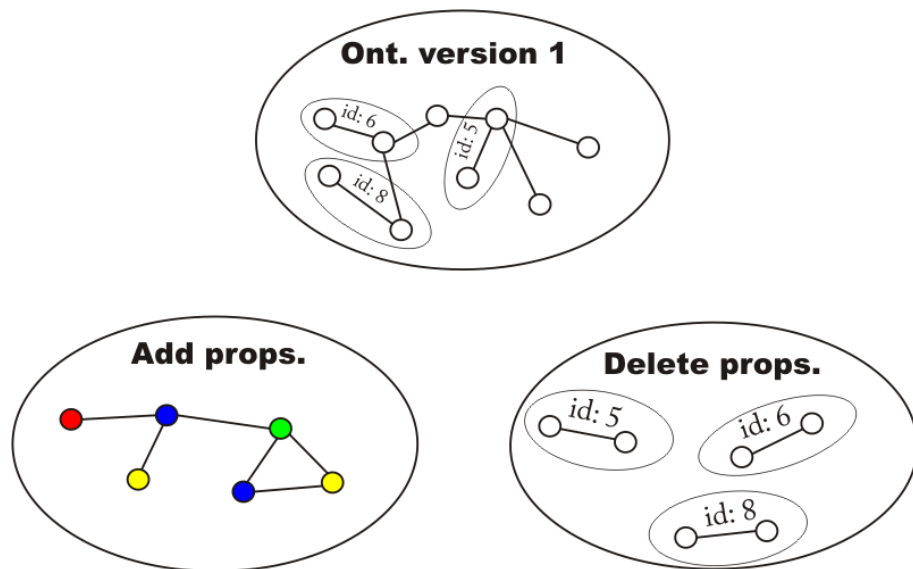
4.3.2 Model zarządzania wersjami

Głównym zadaniem systemu OCS jest składowanie i pozyskiwanie ontologii. W celu zapewnienia elastyczności metody zapisu wiedzy, zdecydowano się na przyjęcie postaci trójkowej, bazującej na modelu opisu zasobów, przedstawionym w rozdziale 2.3.1. Taki sposób zapisu wiedzy nie wymusza od użytkownika dostosowania się do konkretnego dialektu OWL, gdyż w rzeczywistości języki z rodziny OWL są rozszerzeniem modelu RDF. W związku z tym ontologie mogą być zapisane w dowolnym dialekcie OWL, co więcej nawet, w ich rozszerzeniu. Zastosowanie modelu RDF ma także znaczący wpływ na optymalizację czasu dostępu do ontologii. Wpływa również na znaczne uproszczenie mechanizmów wersjonowania ontologii.

Model zarządzania wersjami ontologii w systemie OCS zakłada istnienie następujących elementów:

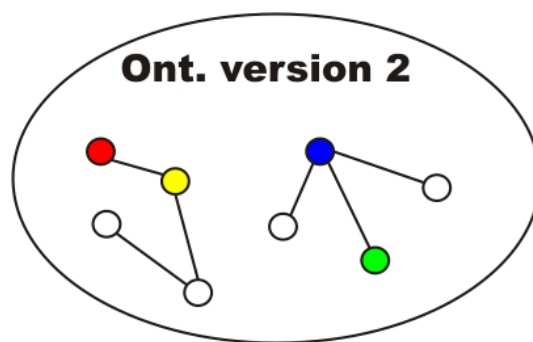
- **bazowa ontologia** - podstawowy element systemu reprezentujący pojedynczą ontologię, będącą modelem pewnej dziedziny. Każda ontologia może mieć wiele wersji. Ontologia „matka” jest ontologią bazową. Ontologia bazowa jest identyfikowana w systemie za pomocą identyfikatora BaseURI (bazowe URI). Identyfikator ten jest typu *String*,
- **ontologia wersjonowana** - to każda kolejna wersja ontologii bazowej. Każda taka ontologia składa się z trójek RDF. Do ontologii wersjonowanej mogą być dołączone propozycje zmian, reprezentujące działania użytkownika. Każda wersjonowana ontologia jest identyfikowana w systemie za pomocą identyfikatora VersionedURI. Identyfikator ten jest w rzeczywistości konkatenacją bazowego URI oraz informacji o wersji i podwersji. Wersja i podwersja to identyfikatory typu *Integer*.
- **propozycja zmiany** - to element pochodzący od użytkownika w momencie wykonywania zmian na ontologii. Możemy wyróżnić dwa rodzaje propozycji zmian: propozycja dodania trójki i propozycja usunięcia trójki. Każda propozycja dodania trójki jest trójką RDF, natomiast propozycja usunięcia trójki wskazuje identyfikator trójki do usunięcia z danej wersji ontologii. Należy tutaj dodać, że zastosowanie

modelu trójkowego nie wymaga wprowadzenia trzeciego rodzaju propozycji zmiany (propozycja edycji), gdyż w rzeczywistości taka propozycja składa się z ciągu: propozycja usunięcia trójki - propozycja dodania nowej trójki.



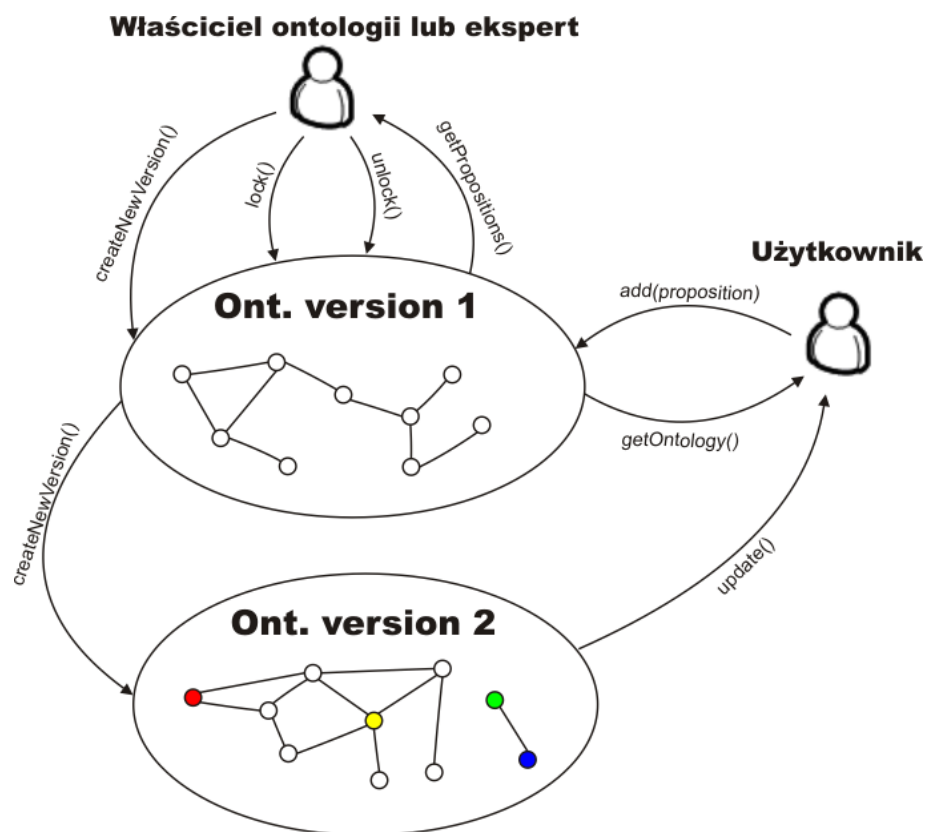
Rysunek 4.3: Proces wprowadzania zmian do ontologii.

W momencie korzystania z systemu użytkownik może wprowadzać zmiany do wersjonowanej ontologii. Zmiany te odnotowywane są w postaci propozycji zmian - dodania nowych trójek, bądź usunięcia trójek. Na rysunku 4.3 przedstawiono proces edycji ontologii o nazwie „Ont. version 1”. Każda propozycja zmiany jest weryfikowana, dzięki czemu unika się sytuacji, w których powstają niepożądane zmiany będące efektem wandalizmu, bądź nieumyślnej zmiany. Kolorowe trójki na rysunku obrazują propozycje dodania nowych elementów do ontologii. Natomiast symbole postaci „id:7”, obrazują identyfikatory trójek, które powinny być usunięte z danej wersji ontologii. Nad całym procesem wprowadzania zmian do ontologii czuwać mają uprzywilejowani użytkownicy: **właściciel ontologii** wraz **ekspertem dziedzinowym**. Użytkownicy ci, mają prawo przeglądać wszystkie propozycje zmian i decydować o ich przyjęciu, bądź odrzuceniu. W wyniku działań właściciela i eksperta otrzymaliśmy kolejną wersję ontologii „Ont. version 1” - „Ont. version 2”. Została ona przedstawiona na rysunku 4.4. Należy zauważyć, że w wyniku tworzenia nowej wersji ontologii nie wszystkie propozycje zmian zostały zatwierdzone. Część z nich została przyjęta, część odrzucona.



Rysunek 4.4: Ontologia po wprowadzeniu zmian.

Powyższy model zarządzania ontologiami został zrealizowany poprzez zdefiniowanie jednolitego interfejsu programistycznego, uwzględniającego mechanizmy tworzenia nowych wersji, synchronizacji, a także blokowania dostępu do ontologii. Interfejs API został zawarty w bibliotece programi-



Rysunek 4.5: Edycja ontologii - widok z poziomu użytkownika.

stycznej, której załączenie w dowolnej aplikacji *Java* włącza funkcjonalność

umożliwiającą dostęp do ontologii. Diagram 4.5 obrazuje sposób wykorzystania API zdefiniowanego w ramach systemu OCS, na podstawie procesu edycji przykładowej ontologii. Zwykły użytkownik (ten, który ma konto w systemie) ma uprawnienia do pobierania dowolnej ontologii i dodawania propozycji zmian. W przypadku powstania nowej wersji ontologii użytkownik może także aktualizować swoją lokalną kopię. Uprzywilejowani użytkownicy - w tym przypadku właściciel, bądź ekspert dziedzinowy posiadają uprawnienia pozwalające na przeglądanie propozycji zmian, blokowanie/odblokowanie ontologii oraz tworzenie nowej wersji ontologii.

Rozdział 5

Analiza i projekt systemu

5.1 Wprowadzenie

Poniższy rozdział przedstawia analizę systemową i projekt techniczny edytora ontologicznego dla systemu OCS. Wszelkie założenia i wymagania odnośnie systemu zostały przedyskutowane na Katedrze Architektury Systemów Komputerowych. Część projektowa i implementacyjna została opracowana w zespole sześciuosobowym. Autor tej pracy odpowiedzialny był za stworzenie warstwy prezentacji, której częścią jest edytor ontologiczny. Wykorzystuje on przyjęte API (patrz punkt 4.2), które zostało zawarte w bibliotece programistycznej.

Rozdział ten jest podzielony na następujące sekcje: 5.2 obrazuje podstawowe sposoby wykorzystanie systemu w postaci diagramu przypadków użycia, 5.3 przedstawia projekt techniczny warstwy prezentacji systemu, 5.4 opisuje strukturę plików wykorzystywanych przez edytor.

5.2 Diagram przypadków użycia

5.2.1 Wprowadzenie

Głównym celem tej pracy było przygotowanie warstwy prezentacji dla systemu OCS, która składała się z trzech elementów:

- **edytora ontologicznego** - narzędzia umożliwiającego kooperacyjną edycję ontologii,

- **stron WWW** - pozwalających na pobranie edytora ontologicznego oraz przeprowadzanie czynności administracyjnych, związanych z zarządzaniem systemem,
- **forum internetowego** - pozwalającego na nawiązanie komunikacji pomiędzy użytkownikami systemu w procesie tworzenia i rozwijania ontologii. Na chwilę obecną część API biblioteki, związane z obsługą forum, nie zostało zrealizowane. W związku z tym autor pracy zdecydował się na przygotowanie „szkieletu” forum internetowego, które posłuży jako podstawa w kolejnych wydaniach systemu.

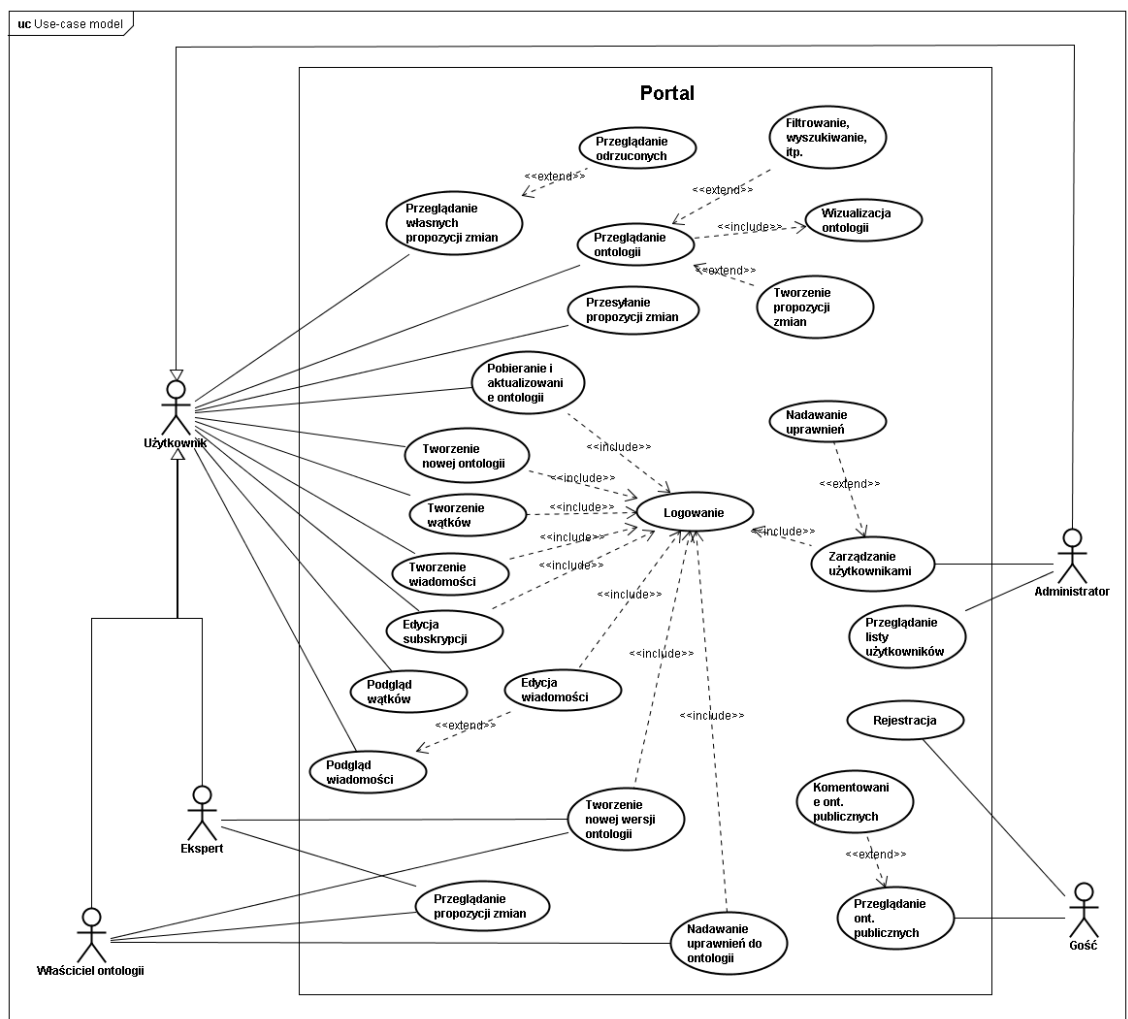
W oparciu o specyfikację wymagań powstał diagram przypadków użycia, obrazujący podstawowe wykorzystania systemu oraz aktorów uczestniczących w ich realizacji. Na rysunku 5.1 przedstawiono model *use-case* dla warstwy prezentacji systemu OCS.

5.2.2 Aktorzy systemowi

Na diagramie przypadków użycia wyróżniono pięciu aktorów. Aktorem stojącym najwyżej w hierarchii dziedziczenia jest „zwykły użytkownik”. Wykonanie jego przypadków użycia jest także możliwe dla aktorów: „ekspert”, „właściciel ontologii” i „administrator”, gdyż są to aktorzy dziedziczący.

Opis aktorów systemowych:

- **Zwykły użytkownik** - to rola reprezentująca wszystkich zarejestrowanych użytkowników systemu, którzy zostali zatwierdzeni przez administratora. Użytkownicy ci mogą pobierać edytor ontologiczny i uczestniczyć w edycji dowolnej ontologii. Mogą oni również uczestniczyć w dyskusjach toczonych na forum internetowym. Każdy taki użytkownik może stworzyć ontologię, wtedy staje się jej właścicielem.
- **Ekspert** - to rola rozszerzająca rolę „zwykły użytkownik” o możliwości tworzenia nowych wersji ontologii, po zatwierdzeniu, bądź odrzuceniu zmian. Ekspert to w rzeczywistości specjalista dziedzinowy - osoba dobrze znającą modelowaną dziedzinę tak, że jego wkład w proces edycji jest istotny. Ekspertem danej ontologii staje się zwykły użytkownik, który został wyznaczony przez właściciela ontologii.
- **Właściciel ontologii** - to rola reprezentująca „zwykłego użytkownika”, który stworzył nową ontologię. W momencie stworzenia staje się



Rysunek 5.1: Diagram przypadków użycia dla systemu OCS.

jej właścicielem. Wraz z ekspertem uczestniczy w procesie weryfikacji zmian wprowadzanych przez innych użytkowników do ontologii. Może przyczynić się do stworzenia nowej wersji ontologii.

- **Administrator** - to rola reprezentująca użytkownika mającego uprawnienia administracyjne w systemie. Administrator zarządza użytkownikami systemu oraz nadaje im odpowiednie uprawnienia. Administrator uczestniczy także w procesie weryfikacji użytkowników, którzy po zarejestrowaniu nie mogą korzystać z systemu.
- **Gość** - to rola reprezentująca użytkownika anonimowego, mającego tylko dostęp do wersji demonstracyjnej systemu. Każdy taki użytkow-

nik może przeglądać ontologie publiczne, jak również je komentować.

5.2.3 Opis przypadków użycia

Nazwa przypadku użycia: Rejestracja.

Aktorzy inicjujący: Gość.

Opis: W momencie, gdy użytkownik nie posiada konta w systemie, może korzystać wyłącznie z wersji demonstracyjnej aplikacji, bądź się zarejestrować. Rejestracja polega na podaniu danych osobowych, które podlegają weryfikacji przez administratora systemu. Gdy administrator zatwierdzi gościa, staje się on zwykłym użytkownikiem systemu.

Nazwa przypadku użycia: Przeglądanie własnych propozycji zmian.

Aktorzy inicjujący: Użytkownik.

Opis: Użytkownik systemu jest w stanie przeglądać utworzone przez siebie propozycje zmian, które zostały przyjęte w nowej wersji ontologii.

Nazwa przypadku użycia: Przeglądanie odrzuconych propozycji zmian.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia rozszerzający przypadek „przeglądanie własnych propozycji zmian”, dotyczący możliwości przeglądania propozycji zmian, które zostały odrzucone w momencie tworzenia nowej wersji ontologii przez właściciela ontologii, bądź przez eksperta dziedzinowego.

Nazwa przypadku użycia: Przeglądanie ontologii.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia umożliwiający przeglądanie elementów składowych ontologii. Wykonanie tego przypadku użycia jest możliwe w momencie wykonania przypadku „wizualizacja ontologii”.

Nazwa przypadku użycia: Filtrowanie, wyszukiwanie ontologii.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia rozszerzający przypadek „przeglądanie ontologii”, dotyczący możliwości filtrowania, bądź wyszukiwania interesujących elementów ontologii przez użytkownika.

Nazwa przypadku użycia: Wizualizacja ontologii.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia wchodzący w skład przypadku „przeglądanie on-

ologii”, dotyczący przedstawiania ontologii w postaci graficznej: jako grafu relacji, bądź jako drzewa taksonomii.

Nazwa przypadku użycia: Tworzenie propozycji zmian.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia rozszerzający przypadek „przeglądanie ontologii”. Dotyczy możliwości edycji istniejącej ontologii. Każda wprowadzana do systemu zmiana jest reprezentowana w postaci „propozycji zmiany”, która jest weryfikowana przez właściciela ontologii, bądź eksperta dziedzinowego w momencie tworzenia nowej wersji ontologii.

Nazwa przypadku użycia: Przesyłanie propozycji zmian.

Aktorzy inicjujący: Użytkownik.

Opis: Po zakończeniu edycji użytkownik może przesłać swoje propozycje zmian na serwer systemu OCS. Przesłane propozycje zmian będą uwzględniane w momencie tworzenia nowej wersji ontologii.

Nazwa przypadku użycia: Pobieranie i aktualizowanie ontologii.

Aktorzy inicjujący: Użytkownik.

Opis: Użytkownik może pobierać każdą ontologię utworzoną w systemie OCS. W momencie edycji konkretnej ontologii, może powstać jej nowa wersja. W ten sposób lokalna kopia ontologii staje się nieaktualna. W związku z tym użytkownik jest w stanie pobrać aktualną wersję ontologii znajdującą się na serwerze OCS.

Nazwa przypadku użycia: Logowanie.

Aktorzy inicjujący: Użytkownik, ekspert, właściciel ontologii, administrator.

Opis: W momencie wykonywania czynności aktualizujących zawartość bazy danych systemu OCS, użytkownik jest proszony o podanie loginu i hasła w celu przeprowadzenie procesów uwierzytelniania i autoryzacji.

Nazwa przypadku użycia: Tworzenie nowej ontologii.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik systemu jest w stanie stworzyć określoną liczbę nowych ontologii. Liczba dozwolonych ontologii, określana jest przez administratora w czasie zatwierdzania użytkownika w systemie.

Nazwa przypadku użycia: Tworzenie wątków.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik, oprócz tworzenia i edycji ontologii, może uczestniczyć w dyskusjach na forum dotyczących zagadnień ontologii. Powyższy przypadek użycia dotyczy dodawania nowego zagadnienia do forum internetowego.

Nazwa przypadku użycia: Tworzenie wiadomości.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik, oprócz tworzenia i edycji ontologii, może uczestniczyć w dyskusjach na forum dotyczących zagadnień ontologii. Powyższy przypadek użycia dotyczy tworzenia nowych wiadomości w obrębie danego zagadnienia na forum internetowym.

Nazwa przypadku użycia: Edycja subskrypcji.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik może zapisać się na listę subskrybentów danej ontologii. Dzięki temu będzie informowany o pojawieniu się kolejnych wersji ontologii.

Nazwa przypadku użycia: Podgląd wątków.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik systemu może przeglądać wątki w obrębie forum internetowego.

Nazwa przypadku użycia: Podgląd wiadomości.

Aktorzy inicjujący: Użytkownik.

Opis: Każdy użytkownik systemu może przeglądać wiadomości w obrębie forum internetowego.

Nazwa przypadku użycia: Edycja wiadomości.

Aktorzy inicjujący: Użytkownik.

Opis: Przypadek użycia rozszerzający przypadek „podgląd wiadomości”. Dotyczy możliwości edycji własnej wiadomości umieszczonej przez danego użytkownika na forum internetowym.

Nazwa przypadku użycia: Przeglądanie propozycji zmian.

Aktorzy inicjujący: Ekspert, właściciel ontologii.

Opis: Ekspert dziedzinowy wraz z właścicielem ontologii mogą tworzyć nowe wersje ontologii poprzez przeglądanie propozycji zmian. Propozycje te

mogą być zatwierdzone bądź odrzucone.

Nazwa przypadku użycia: Tworzenie nowej wersji ontologii.

Aktorzy inicjujący: Ekspert, właściciel ontologii.

Opis: Ekspert dziedzinowy wraz z właścicielem ontologii mogą uczestniczyć w procesie tworzenia nowej wersji ontologii. Dla każdej ontologii istnieje możliwość pobrania propozycji zmian. Nowa wersja ontologii powstaje poprzez promocję lub odrzucenie części propozycji zmian.

Nazwa przypadku użycia: Nadawanie uprawnień do ontologii.

Aktorzy inicjujący: Właściciel ontologii.

Opis: Właściciel ontologii może uczestniczyć w procesie nadawania uprawnień do ontologii. Dotyczą one wyznaczenia ekspertów dziedzinowych spośród użytkowników systemu. Eksperci, podobnie jak właściciele ontologii, mogą tworzyć nowe wersje ontologii.

Nazwa przypadku użycia: Przeglądanie ontologii publicznych.

Aktorzy inicjujący: Gość.

Opis: Użytkownik niezarejestrowany w systemie może uczestniczyć w przeglądaniu ontologii publicznych w ramach wersji demonstracyjnej systemu.

Nazwa przypadku użycia: Komentowanie ontologii publicznych.

Aktorzy inicjujący: Gość.

Opis: Przypadek użycia rozszerzający przypadek „przeglądanie ontologii publicznych”. Użytkownik niezarejestrowany w systemie może komentować ontologie publiczne, dając tym samym informacje zwrotne dla użytkowników systemu.

Nazwa przypadku użycia: Przeglądanie listy użytkowników.

Aktorzy inicjujący: Administrator.

Opis: Administrator może przeglądać listę wszystkich użytkowników systemu, jak również edytować ich uprawnienia w systemie.

Nazwa przypadku użycia: Zarządzanie użytkownikami.

Aktorzy inicjujący: Administrator.

Opis: Przypadek użycia dotyczący podjęcia wszelkich czynności administracyjnych w systemie. Administrator może blokować użytkowników, akceptować nowych użytkowników, nadawać uprawnienia administracyjne, określać liczbę ontologii do utworzenia.

Nazwa przypadku użycia: Nadawanie uprawnień.

Aktorzy inicjujący: Administrator.

Opis: Przypadek użycia rozszerzający przypadek „zarządzanie użytkownikami”. Dotyczy nadawania uprawnień do korzystania z systemu dla użytkowników nowo zarejestrowanych w systemie.

5.3 Projekt techniczny warstwy prezentacji

5.3.1 Podstawowe koncepcje rozwiązania

Realizacja części przypadków użycia systemu, przedstawionych w punkcie 5.2, wymaga stałego połączenia z serwerem, gdyż aktualizują one bezpośrednio zawartość bazy danych. Te funkcjonalności dotyczą głównie wykonywania czynności administracyjnych. Pozostałe przypadki użycia, *stricte* związane z edycją ontologii, mogą być wykonane na lokalnym komputerze użytkownika. Takie rozwiązanie wynika bezpośrednio z założeń nałożonych na edytor ontologiczny systemu OCS (patrz punkt 4.2).

W trakcie korzystania z edytora ontologicznego, użytkownik może ściągnąć dowolną ontologię z serwera systemu OCS i przystąpić do jej edycji. Na komputerze użytkownika tworzona jest kopia lokalna ontologii i zapisywane są wszystkie dokonywane zmiany. Nie wymaga to połączenia edytora z serwerem. W momencie, gdy użytkownik chce przesłać wszystkie zmiany na serwer, dokonywany jest proces porównania kopii lokalnej z kopią wersji bazowej ontologii. W procesie tym uzyskujemy: zbiór propozycji trójek do dodania do ontologii, a także zbiór identyfikatorów trójek do usunięcia z ontologii. W ten sposób powstają propozycje zmian ontologii. Powyższe rozwiązanie ma następujące cechy:

- edycja ontologii nie wymaga pozostawiania *online*. Użytkownik może wprowadzać zmiany będąc niepołączonym z Internetem, natomiast przesyłanie propozycji zmian wymaga ponownego połączenia z serwerem,
- przetwarzanie związane z porównywaniem ontologii odbywa się na maszynie lokalnej, odciążając tym samym serwer z czasochłonnego porównywania ontologii (aplikacja zrealizowana w oparciu o architekturę „grubego klienta”),

- dzięki temu, że edytor dostępny jest zawsze na stronie domowej systemu OCS, użytkownik może pobrać aktualną wersję edytora w „przezroczysty” dla niego sposób.
- koncepcja działania edytora opiera się na wykorzystaniu w procesie edycji ontologii wszelkiego rodzaju „kreatorów” (ang. *wizard*). To rozwiązanie jest powszechnie znane i pozwala na pozyskanie danych od użytkownika w przyjazny sposób. Sposób realizacji „kreatora” został szerzej przedstawiony w rozdziale 5.3.2.

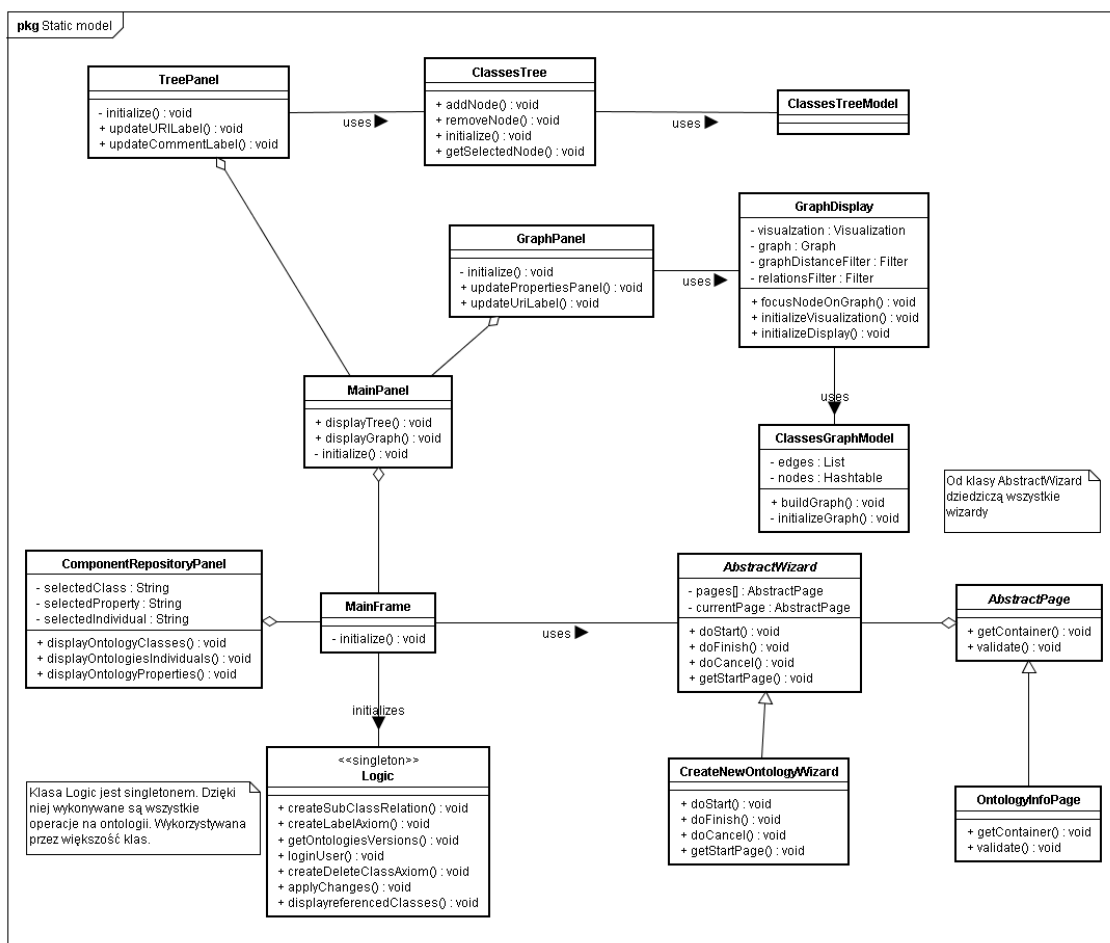
Wszelkie pozostałe czynności: przypisywanie uprawnień do użytkowników, ustalanie ekspertów ontologicznych, subskrypcja ontologii oraz uczestniczenie w dyskusjach na forum internetowym, zostały zrealizowane w oparciu o strony WWW.

5.3.2 Diagram klas edytora ontologicznego

Na rysunku 5.2 przedstawiono poglądowy diagram klas edytora ontologicznego. Uwzględniono w nim najbardziej istotne klasy z punkty widzenia funkcjonalności edytora, aczkolwiek implementacja edytora jest znacznie bardziej rozbudowana i zawiera większą liczbę klas pomocniczych. Edytor zrealizowany jest jako aplikacji oparta o graficzny interfejs użytkownika (ang. *GUI - Graphical User Interface*), w technologii Java/Swing. Technologia ta opiera się o wzorec projektowy MVC, dzięki czemu w aplikacji *Swing* możliwe jest łatwe odseparowanie widoku od modelu danych i akcji pochodzących od użytkownika systemu. Autor niniejszej pracy skorzystał z gotowego komponentu pozwalającego na wizualizację grafów - *prefuse*, który został opisany szerzej w rozdziale 6.

Jak już prędej wspomniano, działanie edytora opiera się o wykorzystanie wszelkiego rodzaju „kreatorów” (ang. *wizard*). Autor pracy zdefiniował prosty interfejs, który pozwala na łatwe tworzenie dowolnego kreatora. Bazując na obserwacjach innych systemów można zauważyć, że każdy kreator składa się z następujących elementów:

- **strony kreatora** - reprezentują formularz, do którego wprowadzane są dane przez użytkownika. Każda strona kreatora reprezentuje oddzielny fragment formularza. Dzięki temu dane pogrupowane są tematycznie,
- **nagłówek kreatora** - panel z informacją o nazwie kreatora oraz o funkcji jaką wykonuje,



Rysunek 5.2: Diagram klas dla edytora ontologicznego.

- **panel nawigacyjny** - dostarcza przycisków pozwalających na przejście do kolejnych stron kreatora oraz na zakończenie jego działania,
- **akcje** - reprezentują wszelkie czynności pochodzące od użytkownika: przejście do kolejnej strony kreatora, zakończenie działania kreatora, wywołanie kreatora, itp.

Powyższe elementy zostały przedstawione na diagramie klas (5.2). Klasa *AbstractWizard* (B.3) reprezentuje podstawowy kreator, natomiast *AbstractPage* (B.4) reprezentuje pojedynczą stronę kreatora. Tworzenie dowolnego kreatora polega na utworzeniu nowej klasy dziedziczącej od *AbstractWizard* i implementacji odpowiednich metod. Utworzenie stron kreatora polega na stworzeniu instancji klasy *AbstractPage* i zdefiniowaniu jej wy-

glądu graficznego. Obsługa przycisków i przekazywanie akcji jest w pełni zautomatyzowane.

Część web'owa aplikacji opiera się w głównej mierze na standardzie JSF, która wchodzi w skład specyfikacji JEE (ang. *Java Enterprise Edition*). Technologia ta jako widok wykorzystuje strony JSP (ang. *Java Server Pages*), które wykorzystują tzw. *managed beans*, czyli komponenty zarządzane. Komponenty te rejestrowane są w odpowiednim pliku konfiguracyjnym aplikacji JSF i odzwierciedlają zawartość formularza, jak również wszelkie akcje, jakie można na nim wykonać.

5.4 Struktura tworzonych plików

Edytor ontologiczny wykorzystuje lokalny system plików w celu przechowywania informacji o aktualnych ustawieniach aplikacji oraz tworzenia kopii ontologii składowanych na serwerze systemu OCS. Poniżej opisano plik konfiguracyjny, odpowiedzialny za zapisywanie aktualnej konfiguracji edytora oraz deskryptor projektu, który zawiera informacje o przetwarzanej ontologii.

5.4.1 Plik konfiguracyjny edytora

Aktualne ustawienia edytora przechowywane są w pliku konfiguracyjnym. Plik ten jest zawsze tworzony przy pierwszym uruchomieniu edytora na danym komputerze, w lokalizacji uzależnionej od zmiennej środowiskowej - %HOMEPATH% (w systemach rodziny Microsoft Windows), lub \$HOME (w systemach rodziny Linux). W obu przypadkach folder ten jest katalogiem domowym użytkownika. Przy każdym uruchomieniu edytora sprawdzane jest czy folder %HOMEPATH%\ocs istnieje. Jeżeli nie - jest tworzony i w pliku *config.xml* zapisywana jest domyślna konfiguracja edytora. Jeżeli katalog %HOMEPATH%\ocs istnieje i plik konfiguracyjny *config.xml* również, wtedy konfiguracja jest odczytywana i zapamiętywana w aplikacji. Plik konfiguracyjny jest typowym plikiem XML, którego znaczniki zostały przedstawione w tabeli 5.1.

Poniżej przedstawiono przykładowy plik konfiguracyjny edytora ontologicznego:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<config>
```

Tabela 5.1: Struktura pliku konfiguracyjnego.

Nazwa znacznika	Opis
<code><config></code>	Znacznik główny pliku, będący jego korzeniem. Nie wnosi żadnych informacji z punktu widzenia konfiguracji.
<code><user></code>	W obrębie tego znacznika przechowywane są informacje o użytkowniku edytora. Znacznik posiada dwa atrybuty: <i>name</i> oraz <i>password</i> , przechowujące informacje wymagane podczas logowania do systemu. Dane te zapisywane są w konfiguracji, gdy użytkownik wybierze opcję „Zapamiętaj mnie” w panelu logowania. Pole <i>password</i> jest zaszyfrowane w celu zapewnienia poufności.
<code><showInferredHierarchy></code>	Znacznik określający czy podczas uruchomienia edytora wyznaczana będzie hierarchia klas w oparciu o wykorzystanie modułu wnioskującego. Autor pracy skorzystał z gotowego narzędzia - <i>Pellet</i> , pozwalającego na wnioskowanie z ontologii OWL DL. Hierarchia ta może być wykorzystana do określenia poprawności konstruowanej ontologii. Znacznik ten zawiera atrybut <i>show</i> , o możliwych wartościach „true”, bądź „false”, określający czy hierarchia będzie wyznaczana.
<code><localeLanguage></code>	Znacznik określający język w jakim będzie uruchamiany edytor. Atrybut <i>language</i> , określa w formie skrótowej aktualnie wybrany język aplikacji.

```

<user name="xh"
      password="38ffd1c6284bad83f9f34643ad26750e" />
<showInferredHierarchy show="false" />
<localeLanguage language="pl" />
</config>

```

Tabela 5.2: Struktura deskryptora projektu.

Nazwa znacznika	Opis
<code><project></code>	Znacznik główny pliku. Posiada atrybut <i>name</i> , określający nazwę projektu.
<code><ontology></code>	Znacznik przechowujący informację o ontologii pobranej z serwera systemu OCS. Posiada trzy atrybuty: <i>baseURI</i> , określający bazowe URI ściągniętej ontologii, <i>version</i> oraz <i>subVersion</i> , określające informacje o wersji i podwersji.
<code><baseOntologyFile></code>	Znacznik przechowujący informację o pliku w którym zapisana została ontologia ściągnięta z serwera systemu OCS. Posiada atrybut <i>fileURI</i> , określający lokalizację pliku z ontologią.
<code><lastModifiedOntologyFile></code>	Znacznik przechowujący informację o pliku w którym została zapisana ontologia po wprowadzeniu zmian przez użytkownika. Posiada atrybut <i>fileURI</i> , określający lokalizację pliku z ontologią.

5.4.2 Deskryptor projektu

Pobranie ontologii z serwera systemu OCS skutkuje utworzeniem projektu. Takie podejście pozwala na utrzymywanie lokalnych kopii ontologii i pracę nad nimi. Dzięki temu użytkownik nie musi być podłączony do Internetu w celu przeprowadzenia edycji pobranych ontologii.

W momencie pobierania ontologii tworzony jest projekt, w którym zapisywana jest ontologia oraz plik przechowujący meta-informacje odnośnie projektu. Domyślnie ontologia zapisywana jest w pliku *base.owl*, natomiast deskryptor projektu w pliku *project.xml*. Plik *base.owl* zawiera ontologię, która jest później wykorzystywana przez edytor do uzyskania listy zmian, dlatego nie powinien być przez nikogo modyfikowany. Podobnie plik *project.xml*, którego modyfikacja może powodować niepoprawne działanie edytora. W tabeli 5.2 przedstawiono opis znaczników występujących w deskrytorze projektu.

Poniżej listing to deskryptor projektu, utworzony dla ontologii pizzy pobranej z serwera systemu OCS:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<project name="pizza" >
  <ontology
    baseURI="importowana" version="1" subVersion="2" />
  <baseOntologyFile
    fileURI="file:/home/jakow/ocs/pizza/base.owl" />
  <lastModifiedOntologyFile
    fileURI="file:/home/jakow/ocs/pizza/zmodyfikowana.owl" />
</project>
```

Rozdział 6

Elementy implementacji

6.1 Wprowadzenie

Poniższy rozdział przedstawia najważniejsze rozwiązania technologiczne wykorzystane podczas implementacji edytora ontologicznego oraz stron WWW. Wybór części rozwiązań, wykorzystanych w trakcie implementacji, wynikał bezpośrednio z wymagań nałożonych na system oraz z doboru architektury. Pozostałe komponenty, ułatwiające implementację systemu, zostały dobrane przez autora pracy w drodze poszukiwań oraz konsultacji z członkami zespołu.

Rozdział ten został podzielony na następujące sekcje: 6.2 przedstawia elementy wykorzystania OWL API, 6.3 ukazuje sposób wdrożenia edytora ontologicznego, 6.4 przedstawia komponent wykorzystany do wizualizacji ontologii oraz 6.5, ukazuje rozwiązania zastosowane podczas tworzenia stron WWW.

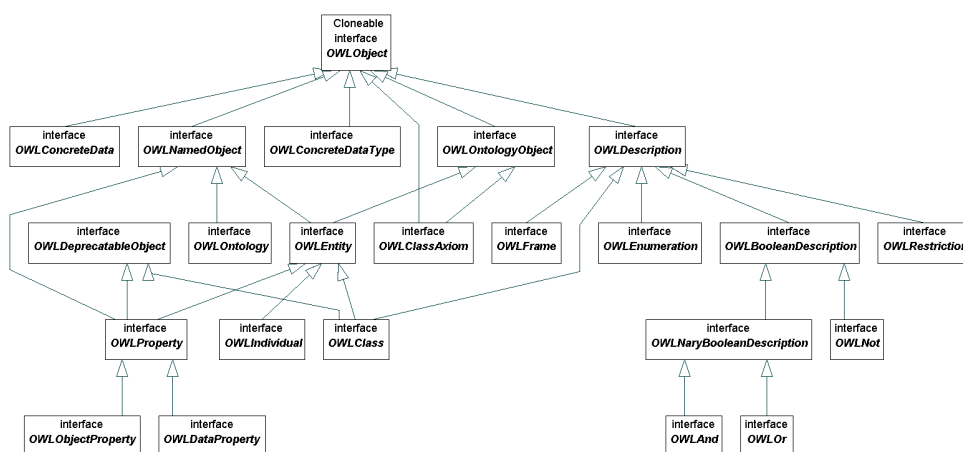
6.2 Integracja z OWL API

OWL API [12, 3] jest interfejsem programistycznym pozwalającym na operowanie na ontologiach OWL. Umożliwia programistom pracę na wysokim poziomie abstrakcji, dzięki realizacji wielu praktycznych, z punktu widzenia aplikacji, funkcjonalności. Pozwala także, na odizolowanie programistów od skomplikowanych zagadnień związanych z serializacją i parsowaniem struktur danych związanych z ontologiami [4, 12]. OWL API dostarcza wspólnej interpretacji wielu języków reprezentacji ontologii (języki

z rodziny OWL, jak również model opisu zasobów RDF). Pozwala to na integrację tego komponentu z narzędziami wykorzystującymi bazy wiedzy, np.: edytory, narzędzia wnioskujące, środowiska agentowe. Wszystkie aplikacje korzystające z OWL API, mogą się ze sobą komunikować przy użyciu wspólnej reprezentacji ontologii. OWL API pokrywa następujące aspekty funkcjonalności związane z dostępem i operowaniem na ontologiach [4]:

- **model** - dostarcza obiektowej reprezentacji ontologii,
- **serializacja** - pozwala na stworzenie konkretnej składni ontologii w oparciu o przechowywany w pamięci model,
- **parsowanie** - pozwala na pobieranie konkretnej reprezentacji ontologii (np.: dokument OWL) i przekształcenie jej do wewnętrznej postaci,
- **edycja** - pozwala na reprezentowanie wszelkich zmian wprowadzonych do ontologii oraz ich zatwierdzanie,
- **wnioskowanie** - pozwala na podłączenie mechanizmów, które uwzględniają formalną semantykę języka reprezentacji ontologii.

Na rysunku 6.1 przedstawiona została hierarchia klas dla opisanego wyżej modułu „model”. Oprócz elementów typowych dla języka OWL, możemy odnaleźć takie, które nie posiadają reprezentacji w specyfikacji OWL, np.: klasa *OWLFrame*. Rozwiązanie przedstawione na powyższym diagramie jest bardzo elastyczne, dzięki czemu istnieje możliwość dostosowania API do zmian zachodzących w specyfikacjach, bądź do własnych potrzeb.



Rysunek 6.1: Uproszczony model OWL API [4].

Implementacja przedstawionych powyżej funkcjonalności OWL API, została pogrupowana w pakiety. Każdy z nich reprezentuje oddzielny aspekt funkcjonalności. Dzięki czemu programista może załączyć tylko wymagane przez swoją aplikację moduły, bez konieczności włączania całej biblioteki. Poniżej przedstawiono pakiety wykorzystane w implementacji edytora ontologicznego, wraz z przykładowym kodem źródłowym.

„Model”

Pakiet ten dostarcza podstawowych metod dostępu do ontologii OWL. W związku z tym możliwe jest pobranie wszelkich istotnych informacji dotyczących ontologii, np.: istnieje możliwość pobrania wszystkich klas, właściwości, bytów które występują w ontologii. Poniższy listing kodu przedstawia sposób pobierania wszystkich podklas dla przykładowej klasy znajdującej się w ontologii.

```
//stworzenie managera
OWLOntologyManager ontologyManager
    = OWLManager.createOWLOntologyManager();
//pobranie ontologii
OWLOntology ontology = ontologyManager
    .loadOntologyFromPhysicalURI(new URI(path));
OWLDataFactory dataFactory = ontologyManager.getOWLDataFactory();

//pobranie przykladowej klasy z ontologii
OWLClass sampleClass = dataFactory
    .getOWLClass(URI.create(classURI));

//pobranie wszystkich podklas dla klasy sampleClass
Set<OWLDescription> subClasses = sampleClass
    .getSubClasses(ontology);

for(OWLClass desc : subClasses) {
    //wypisanie tylko klas nieanonimowych
    //- tych ktore posiadaja URI
    if(!desc.isAnonymous()) {
        System.out.println(desc.asOWLClass().getURI().toString());
    }
}
```

Dodatkowo pakiet ten dostarcza obiektów pomocniczych (narzędziowych), pozwalających na „szybsze” i bardziej czytelne pobieranie interesujących elementów ontologii.

„Change”

Pakiet „model” dostarcza jedynie metod pozwalających na odczyt interesujących informacji z ontologii. Praktycznie nie istnieje żadna możliwość wprowadzenia zmian. Pakiet „change” został zdefiniowany jako rozszerzenie pakietu „model” o możliwość wprowadzania zmian do ontologii. W szczególności pozwala na dodawanie nowych elementów do ontologii, usuwanie istniejących, itp.

Wprowadzanie zmian zostało zrealizowane w oparciu o wzorec projektowy „Rozkaz” (ang. *Command*). Zakłada on istnienie obiektu pomocniczego, który jest odpowiedzialny za akceptowanie zmian. Implementacja tego wzorca projektowego ma „transakcyjny” charakter: albo wszystkie zmiany są zatwierdzane, albo żadna. Poniższy listing obrazuje wprowadzanie zmian do ontologii.

```
/**
 * Metoda dodajaca relacje dziedzicznia dla 2 klas.
 *
 * @param superClassUri uri klasy nadrzecznej
 * @param subClassUri uri klasy podrzecznej
 * @return obiekt reprezentujacy zmiany w ontologii
 */
public OWLAxiomChange createSubClassRelation(String superClassUri
                                           , String subClassUri) {
    OWLClass superClass = dataFactory
        .getOWLClass(URI.create(superClassUri));
    OWLClass newClass = dataFactory
        .getOWLClass(URI.create(subClassUri));

    OWLAxiom axiom = dataFactory
        .getOWLSubClassAxiom(newClass, superClass);
    return new AddAxiom(ontology, axiom);
}

/**
 * Metoda zapisujaca zmiany w ontologii.
```



```

*
* @param changes lista zmian w ontologii
* @throws OWLOntologyChangeException
*/
public void applyChanges(List<? extends OWLOntologyChange> changes)
    throws OWLOntologyChangeException {
    ontologyManager.applyChanges(changes);
}

```

„Inference”

Specyfikacja języka OWL dostarcza nie tylko informacji o składni, lecz także o semantyce języka (patrz rozdział 2): określa formalizm na jakim opiera się dany język, definiuje pojęcia związane z oceną jakości ontologii, itp. Grupę narzędzi pozwalających na uzyskiwanie informacji wynikających z semantyki języka reprezentacji ontologii, nazywamy narzędziami wnioskującymi. Stworzenie takiego mechanizmu nie jest zagadnieniem łatwym do realizacji. Programista musiałby stworzyć automat, który potrafiłby dowodzić twierdzenia logiki opisowej. Dlatego zdecydowano, że w OWL API nie będzie dostarczona żadna implementacja narzędzia wnioskującego. W interfejsie tym dodano jedynie możliwość podłączenia dowolnego „wnioskera”, który musiałby wykorzystywać określone API.

Przykładowym narzędziem wnioskującym jest „Pellet” [26], dla którego zademonstrowano sposób integracji z OWL API.

```

Class<?> reasonerClass = Class
    .forName("org.mindswap.pellet.owlapi.Reasoner");
Constructor<?> con = reasonerClass
    .getConstructor(OWLOntologyManager.class);

OWLReasoner reasoner = (OWLReasoner)con
    .newInstance(ontologyManager);

reasoner.loadOntologies(ontologyManager
    .getImportsClosure(ontology));

//uzyskanie wszystkich podklas do danej klasy
//hierarchia wyznaczona w oparciu o wnioskowanie!!!
Set<OWLClass> subClasses = OWLReasonerAdapter
    .flattenSetOfSets(
        reasoner.getSubClasses(currentClass));

```

```
reasoner.clearOntologies();
```

6.3 Wdrażanie edytora ontologicznego

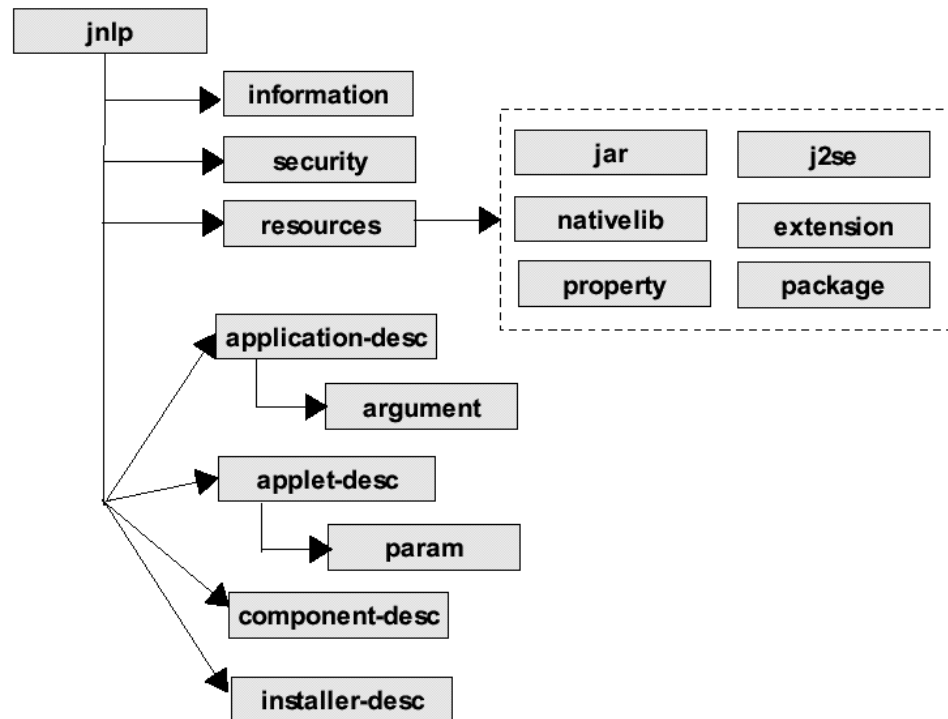
Zgodnie z wymaganiami (4.2), edytor ontologiczny miał być zrealizowany jako narzędzie, dostępne w sieci Web. Platforma Javy dostarcza dwóch alternatywnych dróg pozwalających na pobieranie aplikacji bezpośrednio z sieci: aplety Java oraz Java Web Start. Autor pracy zdecydował się na wykorzystanie technologii Web Start, gdyż w porównaniu z apletami, jest ona bardziej dostosowana do wdrażania aplikacji samodzielnych (ang. *standalone application*).

Interfejs programistyczny Java Web Start jest alternatywą dla apletów. Wykorzystuje on protokół JNLP (eng. *Java Network Launching Protocol*) [16], aby przezroczysto pobierać i instalować aplikacje Javy na lokalnym komputerze. Użytkownik musi jedynie kliknąć łącze instalacji na stronie WWW. Mechanizm Web Start sam przeprowadza instalację: dodaje odpowiednie wpisy do systemu operacyjnego, pobiera wszystkie wymagane zasoby, itp. Zainstalowana aplikacja może później zostać uruchomiona, jak każda inna aplikacja, za pomocą kliknięcia ikony na pulpicie. Stosowane są dla niej zasady bezpieczeństwa i automatycznie sprawdzane uaktualnienia. Java Web Start to postać instalacji klienckiej z zerową administracją, co oznacza, że klient nie musi sam zajmować się instalacją i uaktualnieniami. Aplikacja JNLP może być podpisana lub niepodpisana [22]. Niepodpisane aplikacje Java mają ograniczone uprawnienia, co znacznie zmniejsza ich elastyczność. Każda niepodpisana aplikacja Java, wdrożona jako aplet, bądź Java Web Start, posiada następujące ograniczenia [22]:

- nie może odczytywać ani zapisywać plików z lokalnego systemu,
- może otwierać połączenia sieciowe tylko do serwera, z którego pochodzi,
- nie może uruchamiać nowych procesów na lokalnym komputerze,
- nie może posiadać natywnych bibliotek.

Pakowanie aplikacji jako JNLP jest proste. Wymaga przede wszystkim utworzenia pliku wdrożeniowego JNLP. Następnie na stronie WWW umieszczamy odpowiednie łącze, które uruchamia Web Start. Plik wdrożeniowy

JNLP opisuje sposób w jaki aplikacja Javy ma zostać pobrana i uruchomiona na lokalnym komputerze. Na rysunku 6.2 przedstawiona została struktura pliku JNLP. Posiada on pięć głównych sekcji [16]:



Rysunek 6.2: Struktura pliku wdrożeniowego technologii Java Web Start.

- element `<jnlp>` jest korzeniem pliku JNLP. Posiada meta-informacje dotyczące samego pliku JNLP, np.: wersje specyfikacji protokołu JNLP, przestrzeń nazw aplikacji, itp.,
- element `<information>` dostarcza meta-informacji odnośnie samej aplikacji: właściciel, organizacja, autor, itp. Informacje te wyświetlane są w momencie pobierania aplikacji oraz podczas akceptacji uprawnień,
- element `<security>` opisuje uprawnienia, jakie są wymagane podczas uruchomienia aplikacji,
- element `<resources>` zawiera wszelkie zasoby wymagane przez aplikację, takie jak: klasy Javy, biblioteki JAR, biblioteki z kodem natywnym, itp.,

- ostatnia część pliku składa się z czterech elementów: `<application-desc>`, `<applet-desc>`, `<component-desc>`, `<installer-desc>`. Tylko jeden z nich może być wyrażony w pojedynczym pliku JNLP. Zawierają one szczegółowe informacje dotyczące wdrażanej aplikacji: klasa główna, parametry startowe, itp.

Jak można zauważyć, plik JNLP nie zawiera żadnych danych binarnych. Dostarcza jedynie informacji wymaganych przez maszynę wirtualną w celu pobrania, instalacji i uruchomienia aplikacji.

Poniżej przedstawiono listing pliku JNLP utworzonego podczas wdrażania edytora ontologicznego.

```
<?xml version="1.0" encoding="UTF-8"?>

<jnlp spec="1.0+" codebase="http://ocs.kask.eti.pg.gda.pl">

<information>
  <title>OCS - Ontology creation system</title>
  <vendor>KASK Department, ETI Faculty,
    Gdansk University of Technology
  </vendor>
  <homepage href="index.jsp" />
  <offline-allowed />
</information>

<!-- Edytor wymaga wszelkich uprawnień
  każde archiwum jar musi być podpisane -->
<security>
  <all-permissions/>
</security>

<!-- Klasa główna edytora -->
<application-desc main-class="org.pg.eti.kask.ont.editor.Main" />

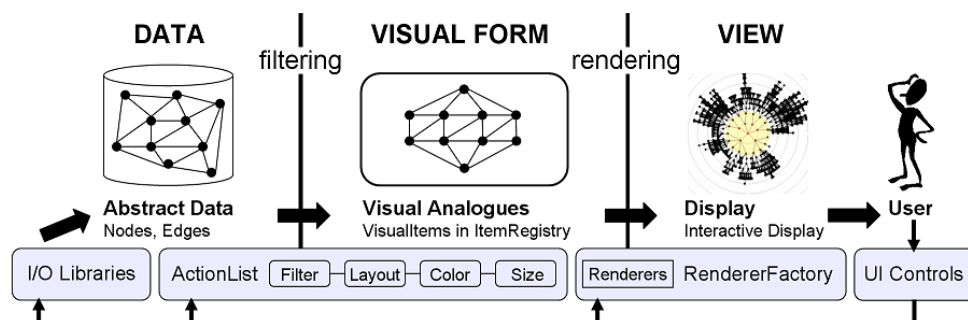
<!-- Wymagane dodatkowe zasoby przez edytor -->
<resources>
  <j2se version="1.6" />
  <jar href="application/ocs-gui.jar" />
  <jar href="application/jlfr-1_0.jar" />
  <jar href="application/antlr-runtime-3.0.jar" />
```

```
<jar href="application/commons-lang-2.2.jar" />
<jar href="application/prefuse.jar" />
<jar href="application/owlapi-api.jar" />
<jar href="application/owlapi-apibinding.jar" />
<jar href="application/owlapi-change.jar" />
<jar href="application/owlapi-debugging.jar" />
<jar href="application/owlapi-dig1_1.jar" />
<jar href="application/owlapi-functionalparser.jar" />
<jar href="application/owlapi-functionalrenderer.jar" />
<jar href="application/owlapi-impl.jar" />
<jar href="application/owlapi-krssparser.jar" />
<jar href="application/owlapi-mansyntaxparser.jar" />
<jar href="application/owlapi-mansyntaxrenderer.jar" />
<jar href="application/owlapi-metrics.jar" />
<jar href="application/owlapi-oboparser.jar" />
<jar href="application/owlapi-owlxmlparser.jar" />
<jar href="application/owlapi-owlxmlrenderer.jar" />
<jar href="application/owlapi-rdfapi.jar" />
<jar href="application/owlapi-rdfxmlparser.jar" />
<jar href="application/owlapi-rdfxmlrenderer.jar" />
<jar href="application/owlapi-util.jar" />
<jar href="application/pellet.jar" />
<jar href="application/commons-logging-1.1.jar" />
<jar href="application/aterm-java-1.6.jar" />
<jar href="application/relaxngDatatype.jar" />
<jar href="application/xsdlib.jar" />
</resources>
</jnlp>
```

6.4 Silnik do wizualizacji grafów

Jednym z głównych elementów edytora ontologicznego jest panel przedstawiający graficzną reprezentację ontologii. Implementacja tego panelu wykorzystuje gotowy komponent, pozwalający na interaktywną wizualizację grafów. Komponent ten nosi nazwę *prefuse* i został pobrany ze strony internetowej: <http://prefuse.org/>. Autorzy silnika udostępnili go na zasadach licencji BSD (ang. *Berkeley Standard Distribution*).

Prefuse jest biblioteką, która może być wykorzystana przy tworzeniu aplikacji opierających się o technologie *infovis* (ang. *information visualization*) [1]. Względem takich narzędzi stawiane są przede wszystkim wymagania dotyczące elastyczności oraz możliwości dostosowania do własnych potrzeb. Autorzy silnika *prefuse* wykorzystali godne uwagi rozwiązanie architektoniczne, które umożliwia rozdzielenia modelu danych od jego wizualnej reprezentacji.



Rysunek 6.3: Architektura silnika *prefuse* [1].

Implementacja przedstawionej na rysunku 6.3 architektury podzielona została na pakiety, które reprezentują odrębne warstwy biblioteki [1, 15]:

- **dane** (*abstract data*) - proces wizualizacji rozpoczyna się od mapowania modelu danych, na model akceptowany przez *prefuse*. Silnik ten dostarcza pewnych predefiniowanych modeli, umożliwiających odwzorowanie danych na graf, bądź drzewo. Istnieje także możliwość rozszerzenia silnika na inne modele zdefiniowane przez użytkownika,
- **filtrowanie** (*filtering*) - jest procesem odpowiedzialnym za odwzorowanie elementów danych na obiekty wizualizowane (*VisualItems*). Takie obiekty posiadają dodatkowe cechy w stosunku do modelu danych, takie jak: kolor, położenie, etykietę, itp. Wszystkie elementy wizualizowane przechowywane są w specjalnie stworzonym rejestrze (*ItemRegistry*). Rejestr ten przechowuje stan elementów wizualizowanych oraz ich mapowanie na model danych,
- **akcje** (*actions*) - reprezentują wszelkie czynności związane z aktualizowaniem widoku. Akcje zmieniają stan wizualizowanych elementów w rejestrze i przeprowadzają m. in.: przekształcenia związane z rozmieszczeniem poszczególnych elementów na ekranie,

- **renderowanie i wyświetlanie** (*rendering and display*) - wizualizowane elementy są rysowane na ekranie przy wykorzystaniu specjalnych obiektów. Prezentacja całej wizualizacji odbywa się poprzez obiekt klasy *Display*, która działaniem przypomina kamerę rejestrującą wizualizowane elementy znajdujące się rejestrze. Obiekt *Display* przyjmuje także wszelkie akcje związane z działalnością użytkownika.

Poniższy listing przedstawia inicjalizację obiektu odpowiedzialnego za wizualizację ontologii w postaci grafu.

```
// inicjalizacja obiektu odpowiedzialnego za
// tworzenie widoku grafu
DefaultRendererFactory rendererFactory
    = new DefaultRendererFactory();
rendererFactory.add(
    new InGroupPredicate(Constants.GRAPH_NODES),
    nodeRenderer);
rendererFactory.add(
    new InGroupPredicate(Constants.EDGE_DECORATORS),
    edgeDecoratorRenderer);
rendererFactory.add(
    new InGroupPredicate(Constants.GRAPH_EDGES),
    edgeRender);

// zarejestrowanie obiektu odpowiedzialnego za tworzenie
// widoku grafu w głównym widoku
visualization.setRendererFactory(rendererFactory);

// zarejestrowanie dekoratora dla krawedzi
Schema edgeDecoratorSchema = PrefuseLib.getVisualItemSchema();
edgeDecoratorSchema.setDefault(VisualItem.INTERACTIVE,
    Boolean.FALSE);
edgeDecoratorSchema.setDefault(VisualItem.TEXTCOLOR,
    ColorLib.gray(0));
edgeDecoratorSchema.setDefault(VisualItem.FONT,
    FontLib.getFont("Tahoma", 8));
visualization.addDecorators(Constants.EDGE_DECORATORS,
    Constants.GRAPH_EDGES, edgeDecoratorSchema);

// krawedzie nie sa interaktywne - nie dziala na nie zadna
```

```
// akcja uzytkownika
visualization.setValue(Constants.GRAPH_EDGES, null,
    VisualItem.INTERACTIVE, Boolean.FALSE);

// ustawienie podswietlonej klasy
VisualItem currentClass = (VisualItem) visualGraph.getNode(0);
visualization.getGroup(Visualization.FOCUS_ITEMS)
    .setTuple(currentClass);
currentClass.setFixed(true);
```

6.5 Komponenty warstwy Web

Warstwa prezentacji systemu OCS składa się nie tylko z edytora ontologicznego, lecz także ze stron WWW i forum internetowego. Zgodnie z wymaganiami oraz przyjętą architekturą (4.2.1), strony WWW zostały zrealizowane przy wykorzystaniu technologii JEE. Technologia ta wyróżnia dwie podstawowe warstwy:

- **warstwa prezentacji** - może zostać zrealizowana w oparciu o serwlety, bądź strony JSP. Specyfikacja JEE dostarcza także rozszerzenie dla powyższych w postaci JSF (ang. *Java Server Faces*). Pozwala ona na tworzenie aplikacji web'owych przy wykorzystaniu wzorca projektowego MVC. Takie rozwiązanie pozwala na dobre odseparowanie modelu danych od widoku. Dzięki temu kod źródłowy aplikacji staje się łatwiejszy w utrzymaniu,
- **warstwa logiki biznesowej** - powinna dostarczać wszelkich metod logiki biznesowej. Może być zrealizowana w oparciu o skalowalne komponenty serwera aplikacji - EJB. Specyfikacja EJB 3.0 definiuje następujące komponenty: sesyjne (ang. *session beans*) i sterowane komunikatami (ang. *message driven beans*)

Warstwa logiki biznesowej została zrealizowana przez innego członka zespołu zajmującego się tworzeniem systemu OCS. Autor niniejszej pracy stworzył warstwę prezentacji (strony WWW), opierając się na specyfikacji JSF. Ponieważ JSF jest specyfikacją, należało pobrać gotowy moduł ją implementujący. Zdecydowano się na przyjęcie implementacji JSF wykonanej przez organizację Apache - MyFaces. Decyzja ta została podjęta ze względu na: uwarunkowania licencyjne (licencja Apache 2.0) oraz dostępność kontrolek rozszerzających funkcjonalności „czystej” specyfikacji JSF. Autor pracy

skorzystał z kontrolek MyFaces Tomahawk, które w znaczący sposób poprawiły wygląd interfejsu użytkownika oraz dostarczyły ciekawych alternatyw w sposobie realizacji poszczególnych funkcjonalności. Do jednych z bardziej ciekawych kontrolek należy: *t:panelTabbedPane*, wyglądem przypominający panel z zakładkami. Równie ciekawą kontrolką jest *t:popup*, która realizuje funkcjonalność okna dialogowego.

Kolejnym elementem części praktycznej tej pracy było stworzenie szkieletu forum internetowego. Zdecydowano się również na wykorzystanie gotowej implementacji forum. Spośród dostępnych na rynku rozwiązań, jednym z bardziej dojrzałych okazało się *jForum*. Pozwala ono między innymi na:

- łatwą instalację silnika forum wraz ze strukturą bazy danych,
- realizację SSO (ang. *Single Sign-On*), dzięki czemu możliwe jest zintegrowanie mechanizmów uwierzytelniania i autoryzacji z inną aplikacją web'ową,
- tworzenie wiadomości i wątków z powszechnie przyjętymi konwencjami panującymi na innych forach internetowych.

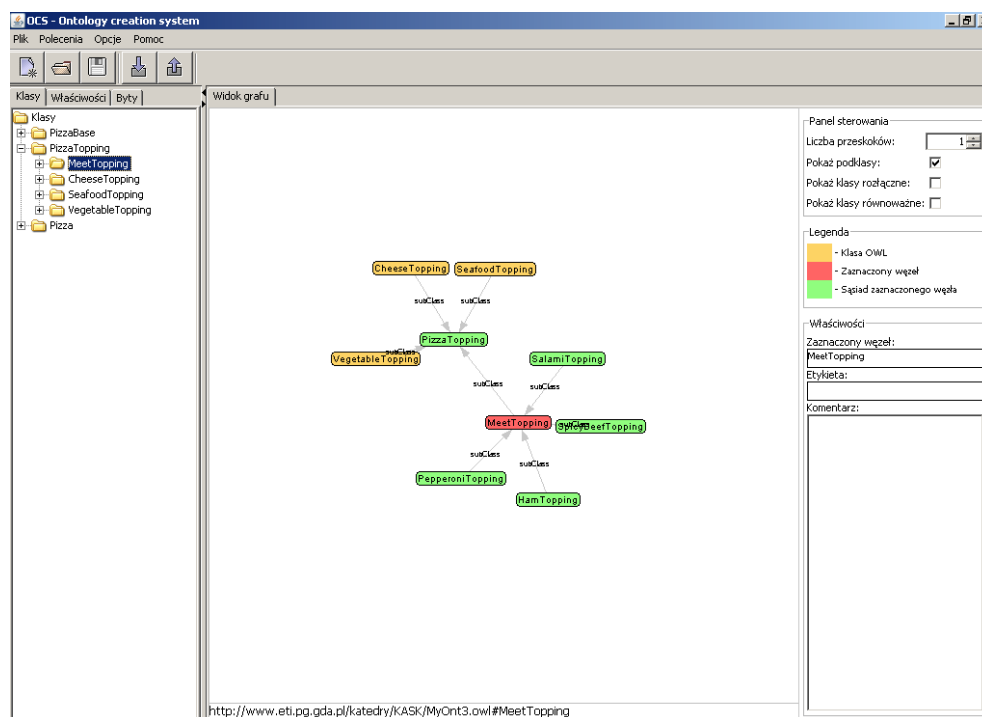
W związku z tym iż część API biblioteki związana z obsługą forum nie została zrealizowana, autor pracy zainstalował aplikację *jForum* na serwerze systemu OCS oraz przystosował ją do obsługi zaakceptowanych użytkowników systemu OCS. W kolejnych wydaniach będzie konieczne lepsze dostosowanie obecnego forum do wymagań względem systemu.

6.6 Wynik implementacji

W wyniku implementacji otrzymano narzędzie pozwalające na kooperacyjną edycję ontologii. Na rysunku 6.4 przedstawiono zrzut ekranu edytora ontologicznego z załadowaną przykładową ontologią.

Wszystkie elementy ontologii dostępne są w repozytorium komponentów. W głównym panelu edytora umieszczony został poglądowy widok ontologii, który umożliwia prześledzenie poszczególnych powiązań pomiędzy klasami ontologii. Autor pracy zdecydował się na podzielenie repozytorium na trzy zakładki (na rysunku widoczne po lewej stronie). W każdej zakładce znajduje się oddzielne drzewo wyświetlające odrębne elementy ontologii:

- **repozytorium klas** - zawiera wszystkie klasy zorganizowane w hierarchię. Hierarchia ta generowana jest na podstawie informacji zawartych wprost w ontologii. Repozytorium to wyświetla wszystkie nieano-



Rysunek 6.4: Zrzut ekranu edytora ontologicznego z przykładową ontologią.

nimowe klasy zawarte w modelu OWL (odpowiadający element OWL - *owl:class*),

- **repozytorium właściwości** - zawiera dwa rodzaje właściwości dla których zakresem są obiekty, bądź wartości. Repozytorium to wyświetla wszystkie nieanonimowe właściwości. Odpowiadające elementy OWL: *owl:dataProperty* dla właściwości, których zakresem są wartości, natomiast *owl:objectProperty* dla właściwości, których zakresem są obiekty,
- **repozytorium bytów** - zawiera instancje klas, inaczej nazywane bytami (ang. *individual*). Repozytorium to wyświetla wszystkie nieanonimowe byty, których odpowiednikiem w OWL jest *owl:individual*.

Dodatkowo autor pracy zdecydował się na wykorzystanie narzędzia wnioskującego Pellet [26], które w obecnej wersji służy do generowania hierarchii klas i bytów na podstawie semantyki danej ontologii (menu „Opcje”, pole „Pokaż wywnioskowaną hierarchię”). Takie podejście znacznie ułatwi pracę użytkownikowi, który w każdym momencie rozwijania ontologii może podejrzeć, czy uzyskał pożądane połączenia. Narzędzie wnioskujące może także

posłużyć do sprawdzenia pewnych aspektów jakościowych ontologii, np.: wspomniane wcześniej w rozdziale 2 sprawdzenie spójności ontologii.

Rozdział 7

Przeprowadzone testy systemu

7.1 Wprowadzenie

Poniższy rozdział przedstawia przyjętą metodologię testowania aplikacji. Cały proces testowania został podzielony na dwa etapy:

- testowanie w całym cyklu wytwarzania - wykonywane w celu sprawdzenia poprawności funkcjonowania poszczególnych metod. Autor pracy zdefiniował i przeprowadził szereg testów modułowych. W tym celu wykorzystano gotowe środowisko pozwalające na zautomatyzowanie procesu testowania - *JUnit*,
- testowanie pod koniec pierwszej iteracji - wykonano testowanie funkcjonalne oraz wydajnościowe. Dla obu tych testów zdefiniowane zostały przypadki testowe oraz środowisko testowania.

Poniższy rozdział podzielony jest na następujące sekcje: 7.2 opisuje sposób wykorzystania biblioteki *JUnit* w testach jednostkowych, 7.3 przedstawia przeprowadzone testy funkcjonalne oraz opisuje ich wyniki, 7.4 przedstawia przeprowadzone testy wydajnościowe systemu oraz krótką dyskusję wyników.

7.2 Testy modułowe

Większość funkcji biznesowych edytora ontologicznego wykonywana jest w specjalnie wyznaczonej do tego klasie *Logic* (B.2). Odpowiedzialna jest ona, między innymi, za wywoływanie funkcji z biblioteki komunikacyjnej,

która pobiera odpowiednie informacje z serwera OCS. Działanie tych funkcji jest krytyczne z perspektywy biznesowej aplikacji. Dlatego też autor pracy zdecydował się na „obłożenie” głównych metod testami jednostkowymi.

Testowanie modułowe w Javie można wykonywać przy użyciu specjalnie do tego zaprojektowanej biblioteki *JUnit*. Biblioteka ta pozwala na zautomatyzowanie procesu testowania oraz na tworzenie zbiorczych raportów. W ten sposób, po każdej aktualizacji biblioteki komunikacyjnej następuje testowanie, które wskazuje potencjalnie niedziałające funkcje.

Poniższy listing przedstawia przykładowy test jednostkowy zrealizowany poprzez użycie biblioteki *JUnit*:

```
@Test
public void getOntologyVersions() throws Exception {
    Logic logic = Logic.getInstance();
    logic.loginUser("xh", "tajne");
    List<BaseURI> ontologies = logic.getOntologiesInfo();
    Assert.assertNotNull(ontologies);
    Assert.assertTrue(ontologies.size() > 0);
    BaseURI baseURI = ontologies.get(0);
    System.out.println(baseURI.getBaseURI());
    List<VersionedURI> versionedURIs = logic.getOntologyVersions(baseURI);
    Assert.assertNotNull(versionedURIs);
    Assert.assertTrue(versionedURIs.size() > 0);
    System.out.println(versionedURIs.size());
}
```

Samo testowanie wykonywane jest poprzez wykorzystanie klasy *Assert*, która udostępnia szereg metod przeprowadzających różne testy, np.: instrukcja *Assert.assertNotNull(ontologies)*; sprawdza, czy zmienna *ontologies* jest różna od wartości *null*. W momencie, gdy *ontologies* jest pusta (*null*), wyrzucany jest odpowiedni wyjątek, natomiast *JUnit* informuje użytkownika o niepowodzeniu testu.

7.3 Testy funkcjonalne

Ten rodzaj testowania dotyczy sprawdzania oprogramowania pod względem zgodności z dokumentem specyfikacji wymagań. Testy funkcjonalne sprawdzają realizację poszczególnych funkcjonalności. Poniżej zdefiniowano przypadki testowe oraz skomentowano ich wyniki.

7.3.1 Specyfikacja i wyniki testów

W ramach testowania funkcjonalnego przeprowadzono testy mające na celu odpowiedź na pytanie, czy system realizuje wyspecyfikowane funkcjonalności. W tym celu zdefiniowano zestaw przypadków testowych, których realizację zweryfikowano podczas wykorzystania systemu. W tabeli 7.1 opisano przypadki testowe oraz przedstawiono wyniki testowania.

Tabela 7.1: Przeprowadzone testy funkcjonalne.

Opis testu	Wyniki
Rejestracja użytkowników	Zrealizowano w postaci formularza rejestracyjnego.
Edycja uprawnień użytkowników.	Zrealizowano w postaci listy użytkowników z możliwością dostępu do formularza edycji.
Zalogowanie użytkownika.	Dostępne przez formularz logowania na stronie WWW oraz w edytorze ontologicznym.
Zalogowanie użytkownika przed akceptacją przez administratora.	Zakończona niepowodzeniem.
Zalogowanie użytkownika po akceptacji przez administratora.	Zakończona sukcesem.
Utworzenie ontologii.	Dostępne z poziomu edytora po wywołaniu kreatora tworzenia nowej ontologii.
Utworzenie listy ekspertów do ontologii.	Dostępne z poziomu stron WWW, po wejściu na stronę z listą ontologii, w zakładce „moje ontologie”.
Dodanie subskrypcji do ontologii.	Dostępne z poziomu stron WWW, po wejściu na stronę z listą ontologii, w zakładce „wszystkie ontologie”.
Pobranie ontologii poprzez zapisanie jej na dysk	Dostępne z poziomu edytora, po wywołaniu kreatora importu ontologii.
Dodanie nowych klas, edycja klas, usuwanie klas.	Funkcje dostępne z poziomu edytora, w menu kontekstowym w repozytorium komponentów.
Dodanie nowych właściwości, edycja właściwości, usuwanie właściwości.	Funkcje dostępne z poziomu edytora, w menu kontekstowym w repozytorium komponentów.

Dodanie nowych bytów, edycja bytów, usuwanie bytów.	Funkcje dostępne z poziomu edytora, w menu kontekstowym w repozytorium komponentów.
Zapisanie zmienionej ontologii na dysk.	Dostępne z poziomu edytora, po wywołaniu okna dialogowego do zapisu ontologii na dysk.
Odczytanie ontologii z dysku i wysłanie propozycji zmian na serwer.	Dostępne z poziomu edytora, po wczytaniu ontologii (okno dialogowe „otwórz”) wywołanie kreatora eksportu zmian na serwer.
Utworzenie nowej ontologii poprzez zatwierdzenie części propozycji zmian i odrzucenie pozostałych.	Dostępne z poziomu edytora po wywołaniu kreatora tworzenia nowej wersji, znajdującego się w menu głównym edytora.
Przeglądanie propozycji zmian.	Dostępne z poziomu edytora po wywołaniu okna dialogowego ze zmianami, znajdującego się w menu głównym edytora.

7.4 Testy wydajnościowe

Testy wydajnościowe mogą dotyczyć wielu różnych scenariuszy. Za ich pomocą można określić m.in.: zachowanie systemu w momencie dużego obciążenia. Można również określić, jak dużą liczbę użytkowników jest w stanie obsłużyć system. W niniejszej części starano się określić czas odpowiedzi edytora ontologicznego w momencie wywoływania najważniejszych funkcji biznesowych.

7.4.1 Specyfikacja i wyniki testów

W celu przeprowadzenia testów, w pierwszej kolejności zdefiniowano przypadki testowe oraz ustalono środowisko testowania aplikacji. Autor wyróżnił następujące przypadki testowe:

1. Określenie średniego czasu serializacji ontologii z wewnętrznego modelu OWL-API na postać trójkową, zapisywaną w bazie danych systemu OCS.
2. Określenie średniego czasu pobierania ontologii o różnych rozmiarach, z serwera systemu OCS.

3. Określenie średniego czasu importowania ontologii o różnych rozmiarach, na serwer systemu OCS.

Dla pierwszego przypadku testowego środowiskiem testowania był komputer wyposażony w procesor AMD Athlon XP 1700+ (rzeczywisty zegar: 1.47GHz) oraz 1.5GB pamięci RAM. Dla pozostałych przypadków testowych (drugiego i trzeciego), testowano czas przesyłania komunikatów HTTP w sieci LAN 100 Mbit'owej.

Na potrzeby testów wykorzystano przykładowe ontologie pobrane z publicznie dostępnych źródeł. W celu uzyskania miarodajnych wyników testów, wybrano ontologie o różnych rozmiarach, których właściwości zostały przedstawione w tabeli 7.2.

Tabela 7.2: Ontologie użyte w testach.

Nazwa ontologii	Liczba trójek	Rozmiar na dysku
beer	179	11.1kB
bhakti-yoga	187	13.65kB
pizza1	213	24.64kB
pizza2	775	77.26kB
aminokwasy	1 195	97kB
cosmo	22 477	3.18MB
emap	96 153	7.19MB
human-dev-anat	125 075	8.66MB
mammalian-phenotype	130 790	10.22MB
molecular-function	305 946	22.43MB

„Czas serializacji”

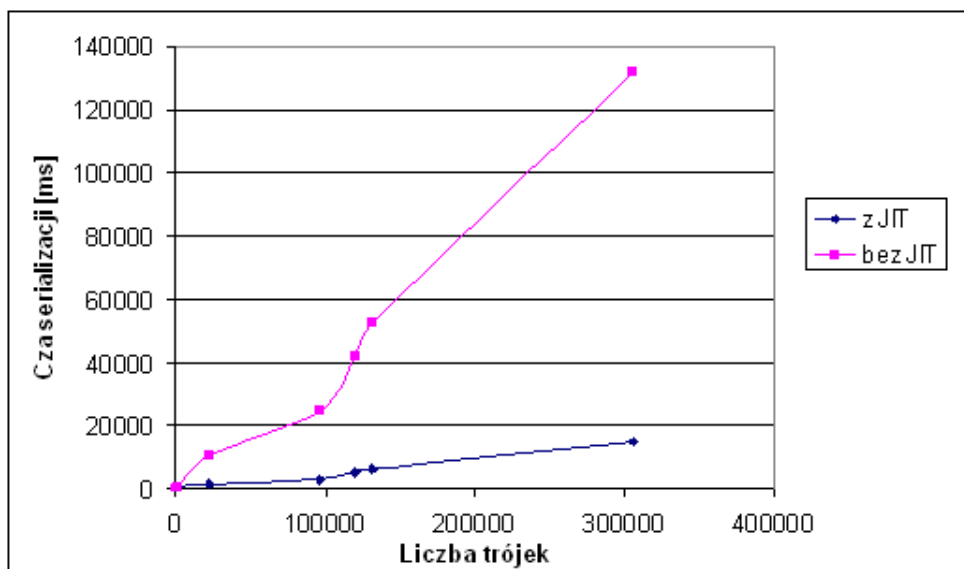
Maszyna wirtualna Javy posiada specjalny mechanizm przyspieszający wykonanie kodu klas (ang. *bytecode*). Mechanizm ten to kompilator JIT (ang. *Just In Time Compiler*). Jego działanie opiera się na kompilacji części kodu klas Javy do kodu natywnego, przez co znacznie skraca się czas wykonania aplikacji. Jednak stosowanie tego mechanizmu tworzy narzut związany z czasem inicjalizacji JIT. Także samo jego stosowanie wpływa na określenie względnego czasu wykonania aplikacji, gdyż nie wiadomo jaki procent kodu został przekompilowany przez JIT. W związku z tym, przeprowadzono dwa testy: jeden z włączonym mechanizmem JIT (domyślne uruchomienie maszyny wirtualnej), drugi z wyłączonym (opcja *-Xint* podczas uruchamiania maszyny wirtualnej).

Dla każdej ontologii z tabeli 7.2, przeprowadzono po 10 testów i wyznaczony czas uśredniono. Wyniki przedstawiono w tabeli 7.3.

Tabela 7.3: Czas serializacji ontologii do postaci trójkowej.

Nazwa ontologii	Średni czas (z JIT)[ms]	Średni czas (bez JIT)[ms]
beer	140	164
bhakti-yoga	140	160
pizza1	172	188
pizza2	250	468
aminokwasy	266	484
cosmo	1218	10532
emap	2966	24358
human-dev-anat	5172	41922
mammalian phenotype	6094	52108
molecular function	14938	131812

Wnioski: Dla małych ontologii (z niewielką liczbą trójek), czas serializacji nieznacznie się zmienia. Wraz ze wzrostem liczby trójek, znacząco ujawnia się mechanizm optymalizacji maszyny wirtualnej. Najprawdopodobniej spowodowane jest to występowaniem w kodzie programu pętli, których wykonanie w głównej mierze wpływa na całkowity czas wykonania testu. Pętle są natomiast promowane przez kompilator JIT w procesie optymalizacji. Na



Rysunek 7.1: Wykres średniego czasu serializacji do postaci trójkowej.

rysunku 7.1 przedstawiono wykresy czasów serializacji dla maszyny wirtu-

alnej z włączonym oraz z wyłączonym mechanizmem JIT.

„Czas pobierania ontologii”

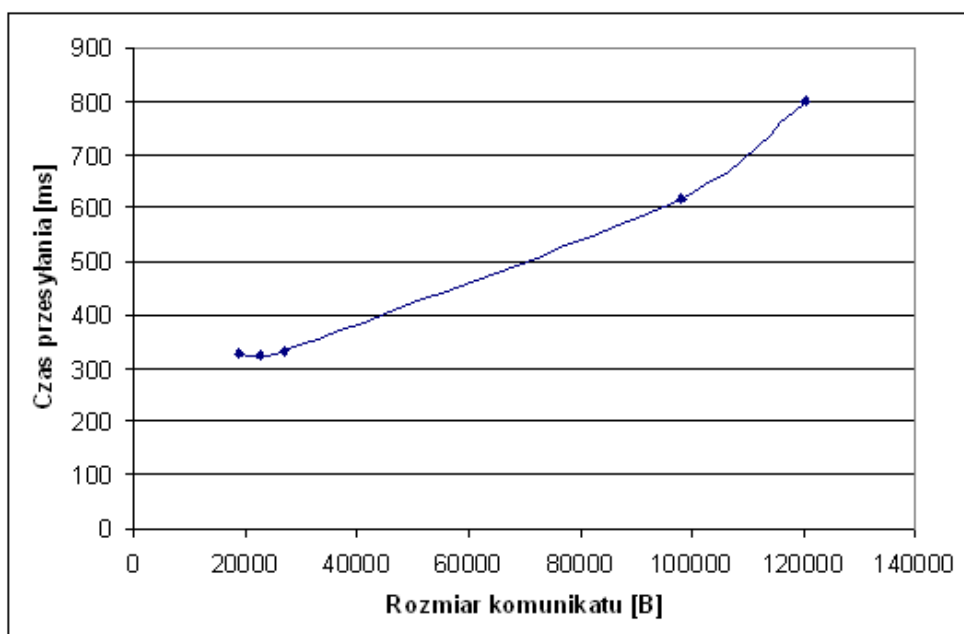
Test ten miał na celu określenie czasu odpowiedzi edytora w momencie pobierania ontologii z serwera OCS. Pobieranie ontologii odbywa się za pośrednictwem biblioteki, która „opakowuje” całość komunikacji zachodzącej pomiędzy edytorem a serwerem. W rzeczywistości, biblioteka ta wykorzy-

Tabela 7.4: Rozmiary komunikatów i średni czas pobierania ontologii z serwera OCS.

Nazwa ontologii	Rozmiar komunikatu	Średni czas przesyłania [ms]
beer	18945B	326
bhakti-yoga	22645B	324
pizza1	27109B	332
pizza2	97896B	618
aminokwasy	120374B	798
cosmo	nie przeprowadzono	nie przeprowadzono
emap	nie przeprowadzono	nie przeprowadzono
human-dev-anat	nie przeprowadzono	nie przeprowadzono
mammalian phenotype	nie przeprowadzono	nie przeprowadzono
molecular function	nie przeprowadzono	nie przeprowadzono

stuje protokół HTTP, w ramach którego przesyłane są odpowiednie komunikaty.

Przed przystąpieniem do testu wykonano import ontologii przedstawionych w tabeli 7.2. Niestety w pierwszym wydaniu systemu przeprowadzanie części testów nie było możliwe. Wynikało to z błędów w bibliotece komunikacyjnej, która była wykonywana przez inną grupę. Na chwilę obecną, autor pracy zaimportował część ontologii z tabeli 7.2, a następnie dokonał testowych pobrań w celu określenia rozmiaru komunikatów podczas pobierania każdej ontologii osobno. Rozmiar komunikatów, przesyłanych pomiędzy edytorem a serwerem systemu OCS, określono przy wykorzystaniu narzędzia *tcpmon*, dostępnego w pakiecie oprogramowania *Apache Axis*. Właściwych testów dokonano w laboratorium, gdzie komputer kliencki połączony jest z serwerem siecią LAN. Dla każdej ontologii dokonano 10 testowych pobrań, a czas pomiaru uśredniono. Wyniki testów zostały zamieszczone w tabeli 7.4 i na rysunku 7.2. Dla ontologii, których import nie powiódł się zaznaczono, że testu nie przeprowadzono.



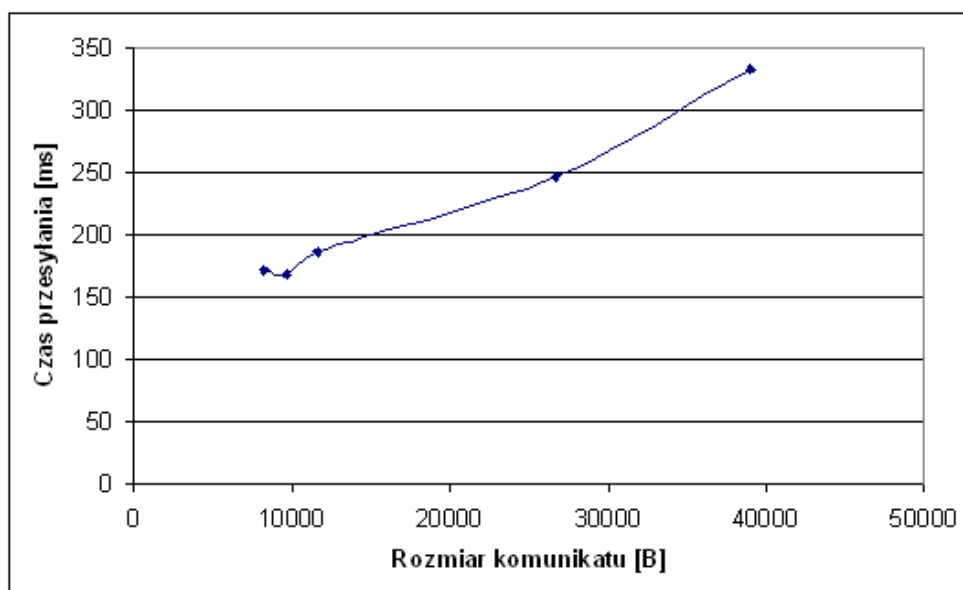
Rysunek 7.2: Wykres średniego czasu pobierania ontologii z serwera OCS.

„Czas importowania ontologii”

Test ten miał na celu określenie czasu odpowiedzi edytora w momencie przesyłania ontologii na serwer systemu OCS. Import ontologii, podobnie jak jej pobieranie, odbywa się również za pośrednictwem biblioteki, która „opakowuje” całość komunikacji.

Tabela 7.5: Rozmiary komunikatów oraz średni czas przesyłania na serwer OCS.

Nazwa ontologii	Rozmiar komunikatu	Średni czas przesyłania [ms]
pizza1	8267B	172
beer	9627B	168
bhakti-yoga	11668B	186
pizza2	26673B	246
aminokwasy	39035B	332
cosmo	nie przeprowadzono	nie przeprowadzono
emap	nie przeprowadzono	nie przeprowadzono
human-dev-anat	nie przeprowadzono	nie przeprowadzono
mammalian phenotype	nie przeprowadzono	nie przeprowadzono
molecular function	nie przeprowadzono	nie przeprowadzono



Rysunek 7.3: Wykres średniego czasu przesyłania ontologii na serwer OCS.

Przed przystąpieniem do testu, wykonano import każdej ontologii z tabeli 7.2, w celu dokonania pomiaru przesyłanych komunikatów. W związku z błędami w bibliotece komunikacyjnej, część importów ontologii nie powiodła się. Podobnie jak w teście dotyczącym określenia czasu pobierania ontologii, pomiary komunikatów dokonano przy użyciu narzędzia *tcpmon*. Właściwe testy polegały na wykonaniu serii 10 importów, a czas pomiaru uśredniono. Wyniki testów zostały zamieszczone w tabeli 7.5 i na rysunku 7.3.

Rozdział 8

Podsumowanie

Tworzenie systemu OCS było dla autora tej pracy źródłem bezcennych doświadczeń, które zostaną wykorzystane w przyszłości. Nieocenioną wartość miały również spotkania grupy projektowej, na których przedyskutowano ciekawe rozwiązania, mające odzwierciedlenie w architekturze systemu.

Autor pracy zrealizował postawione przed nim zadania, w wyniku których powstała pierwsza wersja systemu OCS. W pełni zaimplementowano warstwę prezentacji w postaci stron WWW, utworzono szkielet forum internetowego i przede wszystkim, stworzono prototyp edytora ontologicznego.

Cała funkcjonalność systemu została przetestowana i wdrożona. System jest dostępny pod adresem internetowym: <http://ocs.kask.eti.pg.gda.pl>. Przed następcami autora postawione będzie zadanie ulepszenia edytora ontologicznego, co w przyszłości zaowocuje jego drugą wersją. Już w tym momencie autor pracy zauważył możliwości w zakresie rozbudowania systemu:

- ulepszenie panelu z wizualizacją ontologii tak, aby umożliwić edycję z poziomym grafu,
- stworzenie elementów edytora pozwalających na łączenie wielu ontologii,
- udostępnienie możliwości edycji dla wszystkich elementów ontologii, zdefiniowanych w OWL API,
- dostosowanie forum internetowego do potrzeb jego użytkowników.

Obecnie, wykorzystanie biblioteki OWL API podczas implementacji systemu, wydaje się bardzo rozsądnym rozwiązaniem, gdyż w przyszłości bi-

blioteka ta może stać się standardem. Wielu twórców edytorów (np.: Protégé 3.2) już teraz zdecydowało, że kolejne wersje edytora zostaną „przepisane” na OWL API. Dlatego można śmiało stwierdzić, że system OCS jest innowacyjny. Po dopracowaniu istniejących mechanizmów i wprowadzeniu nowych funkcjonalności do edytora, można będzie stwierdzić, że stanie się on narzędziem, które będzie mogło konkurować z innymi, dostępnymi na rynku edytorami ontologicznymi.

Bibliografia

- [1] Jeffrey Heer, Stuart K. Card and James A. Landay. *prefuse: a toolkit for interactive information visualization*. *CHI 2005*, 2005.
- [2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook. Theory, implementation and applications*. Cambridge University Press, 2004.
- [3] Sean Bechhofer. *Programming to the OWL API: Introduction*. University of Manchester, 2007.
- [4] Sean Bechhofer, Raphael Volz, and Phillip Lord. *Cooking the Semantic Web with the OWL API*. 2nd International Semantic Web Conference, ISWC, Sanibel Island, Florida, 2003.
- [5] Dave Beckett and Brian McBride. *RDF Syntax Specification*. <http://www.w3.org/TR/rdf-syntax-grammar/>, 2004.
- [6] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web*. Scientific American, 2001.
- [7] Mike Dean and Guus Schreiber. *OWL Web Ontology Language Reference*. <http://www.w3.org/TR/owl-ref/>, 2004.
- [8] Bruce Eckel. *Thinking in Java. Edycja polska. Wydanie IV*. Helion, 2006.
- [9] David Geary and Cay S. Horstmann. *JavaServer Faces. Wydanie II*. Helion, 2008.
- [10] Krzysztof Goczyla and Teresa Zawadzka. *Ontologie w Sieci Semantycznej*. Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, 2006.

- [11] Patrick Graessle, Henriette Baumann, and Philippe Baumann. *UML 2.0 w akcji. Przewodnik oparty na projektach*. Wydawnictwo Helion, Gliwice, 2006.
- [12] Matthew Horridge, Sean Bechhofer, and Olaf Noppens. *Igniting the OWL 1.1 Touch Paper: The OWL API*. OWLED 2007, 3rd OWL Experienced and Directions Workshop, 2007.
- [13] Ian Horrocks. *OWL: a Description Logic Based Ontology Language*. School of Computer Science, University of Manchester, 2005.
- [14] <http://pl.wikipedia.org/wiki/Synergia>. Synergia.
- [15] <http://prefuse.org/doc/manual/>. Prefuse manual.
- [16] Inc. Java Software A Division of Sun Microsystems. Java network launching protocol & api specification (jsr-56).
- [17] Andrzej Kobylński. *ISO/IEC 9126 - Analiza modelu jakości produktów programowych*. Konferencja Systemy Wspomagania Organizacji, 2003.
- [18] Henryk Krawczyk and Rafał Mikołajczak. *Internetowy system oceny jakości oprogramowania*. Zeszyty Naukowe Wydziału ETI Politechniki Gdańskiej. Technologie Informacyjne., 2006.
- [19] Frank Manola and Eric Miller. *RDF Primer*. <http://www.w3.org/TR/rdf-primer/>, 2004.
- [20] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. <http://www.w3.org/TR/2004/REC-owl-features-20040210/>, 2004.
- [21] Marvin Minsky. *A Framework for Representing Knowledge*. McGraw-Hill, 1975.
- [22] Patrick Niemeyr and Jonathan Knudsen. *Java. Wprowadzenie*. Wydawnictwo Helion, Gliwice, 2003.
- [23] Nataly F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University, 2000.
- [24] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. *Debugging OWL ontologies*. In Proceedings of the 14th International World Wide Web Conference (WWW-2005), 2005.
- [25] Helena Rasiowa. *Wstęp do matematyki współczesnej*. Wydawnictwo Naukowe PWN, 2003.

- [26] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. *Pellet: A practical OWL-DL reasoner*. Journal of Web Semantics, 2007.
- [27] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. *OWL Web Ontology Language Guide*. <http://www.w3.org/TR/owl-guide/>, 2004.
- [28] Steffen Staab and Rudi Studer. *Handbook on Ontologies*. Springer Verlag, 2004.
- [29] Inc. Sun Microsystems. *The Java EE 5Tutorial*. Sun Microsystems, Inc., 2007.

Dodatek A

Instrukcja obsługi systemu

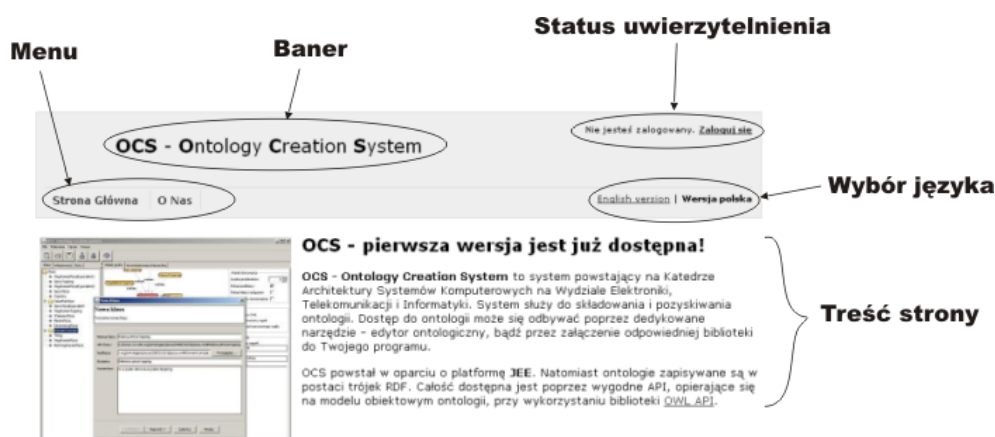
Poniższy rozdział zawiera instrukcje użytkowania systemu OCS. Przedstawiono w nim sposób wykorzystania edytora ontologicznego, jak również stron WWW znajdujących się na serwerze.

A.1 Dostęp przez strony WWW

A.1.1 Struktura strony

System OCS (edytor ontologiczny, forum internetowe oraz strony WWW) dostępny jest pod adresem internetowym: <http://ocs.kask.eti.pg.gda.pl>. W momencie jego wywołania w przeglądarce internetowej, użytkownik zostanie przekierowany na stronę domową systemu. Wygląd stron WWW oparty jest o szablon, który został przedstawiony na rysunku A.1. Wyróżnia on następujące składowe interfejsu:

- **baner** - element przedstawiający logo i nazwę systemu,
- **menu** - pozwala na wywołanie operacji w systemie, np.: pobranie edytora ontologicznego, przejście do forum internetowego, zarządzanie ontologiami, itp.,
- **status uwierzytelnienia** - element wyświetlający informacje o statusie zalogowania użytkownika w systemie,
- **wybór języka** - element pozwalający na wybór innej wersji językowej systemu. W pierwszej wersji systemu dostępne są dwie wersje językowe: polska i angielska,



Rysunek A.1: Główny szablon strony.

- **treść strony** - element przedstawiający treść główną strony. Wyświetlana treść jest zależna od wyboru w menu, dokonanego przez użytkownika.

Dostęp do większości funkcji systemu wymaga zalogowania. Dla użytkownika anonimowego pozostaje możliwość rejestracji, przejrzania wersji demonstracyjnej systemu oraz przeczytania informacji o autorach.

A.1.2 Logowanie i rejestracja w systemie

Rysunek A.2: Formularz logowania.

Po wybraniu opcji „Zaloguj się” znajdującej się w części „status uwierzytelnienia”, pojawia się formularz logowania (rysunek A.2). Użytkownik

nie posiadający konta w systemie musi się zarejestrować, aby móc skorzystać z jego wszystkich funkcji. Proces rejestracji polega na wpisaniu danych osobowych do formularza rejestracyjnego (rysunek A.3) i przesłaniu ich na serwer. Bezpośrednio po zarejestrowaniu się w systemie, nie istnieje możli-

The screenshot shows the 'Rejestracja nowego użytkownika' (New user registration) page of the OCS system. At the top, it says 'OCS - Ontology Creation System' and 'Nie jesteś zalogowany. [Zaloguj się](#)'. Below this is a navigation bar with 'Strona Główna' and 'O Nas', and language options 'English version' and 'Wersja polska'. The main heading is 'Rejestracja nowego użytkownika'. The instructions state: 'Proszę wypełnić **wszystkie pola**. Po zatwierdzeniu przez **Administrатора** możliwe będzie **zalogowanie**.' The form contains five input fields: 'Nazwa użytkownika:', 'Hasło:', 'Powtórz hasło:', 'Imię i nazwisko:', and 'E-mail:'. A 'Wyślij' button is at the bottom right of the form. The footer contains copyright information: '© Copyright 2008 KASK WETI Politechnika Gdańska'.

Rysunek A.3: Formularz rejestracji nowego użytkownika.

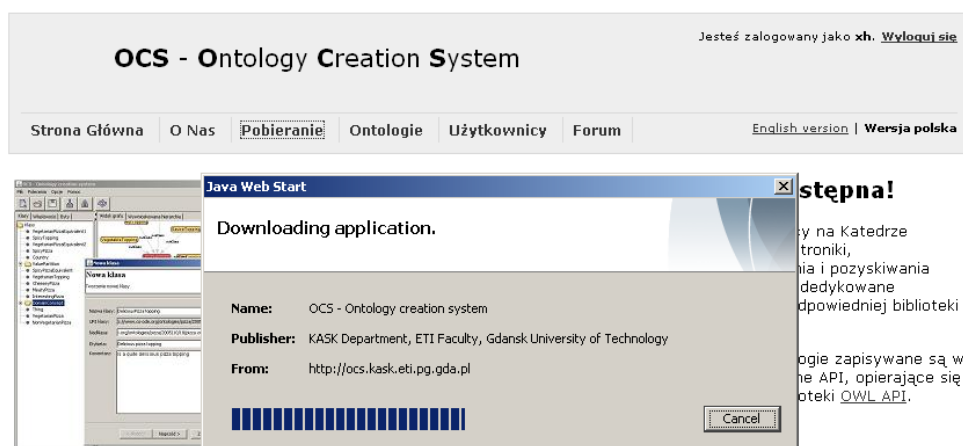
wość zalogowania się. Użytkownik musi oczekiwać na zaakceptowanie jego konta przez administratora systemu. Dopiero, gdy administrator zatwierdzi konto użytkownika, powstanie możliwość zalogowania się do systemu.

A.1.3 Pobieranie edytora ontologicznego

Edytor ontologiczny można pobrać ze strony systemu OCS. W celu ściągnięcia edytora wymagane jest zalogowanie się w systemie. Gdy się powiedzie, użytkownik zostanie przekierowany na stronę domową. Zmianie ulegnie również menu systemowe, które zostanie rozszerzone o funkcje specyficzne dla danego użytkownika. Jedną z tych funkcji jest „Pobieranie”. Kliknięcie na to łącze, powinno zwrócić plik o nazwie *ocs-gui.jnlp*. Otwarcie tego pliku spowoduje automatyczne uruchomienie mechanizmu Java Web Start, który pobierze, zainstaluje oraz uruchomi edytor ontologiczny. Rysunek A.4 przedstawia instalację edytora ontologicznego przez narzędzie Java Web Start.

A.1.4 Zarządzanie ontologiami

Z poziomu stron WWW możliwe jest także zarządzanie uprawnieniami do ontologii. Polega ono na określeniu, którzy użytkownicy systemu są



Rysunek A.4: Pobieranie edytora ontologicznego.

ekspertami w stosunku do dziedziny modelowanej przez daną ontologię. Uprawnienia eksperckie może nadawać właściciel danej ontologii.

W celu wyświetlenia listy ontologii, należy kliknąć łącze „Ontologie” w menu systemu. Na rysunku A.5 przedstawiono zrzut ekranu, po przejściu na stronę zarządzania ontologiami. Strona ta została podzielona na trzy części:

OCS - Ontology Creation System			
Jesteś zalogowany jako xh . Wyloguj się			
Strona Główna	O Nas	Pobieranie	Ontologie
Użytkownicy	Forum	English version	Wersja polska
Lista ontologii			
Wszystkie ontologie		Moje ontologie	
Subskrybowane ontologie			
Bazowe URI		Nazwa ontologii	Subskrypcja Wersje
importowana			Subskrybuj Schowaj
Wersja	Podwersja	Opis wersji	
1	1		
1	2		
pusta			Subskrybuj Pokaż

Rysunek A.5: Zarządzanie ontologiami.

- zakładka „wszystkie ontologie” - wypisuje wszystkie ontologie przechowywane w systemie wraz z informacjami o wersjach i podwersjach. Każdy użytkownik systemu może ustanowić subskrypcję do każdej ontologii przechowywanej w systemie. W tym celu należy kliknąć łącze „Subskrybuj”,

- zakładka „moje ontologie” - wypisuje wszystkie ontologie, których właścicielem jest zalogowany użytkownik. Właściciel ontologii może przypisać ekspertów do swoich ontologii. W tym celu należy kliknąć łącze „Dodaj eksperta ontologii”,
- zakładka „subskrybowane ontologie” - wypisuje wszystkie ontologie, które są subskrybowane przez zalogowanego użytkownika. Istnieje możliwość usunięcia subskrypcji poprzez kliknięcie łącza „Usuń”.

A.1.5 Zarządzanie użytkownikami

Dla administratora systemu przewidziany został panel zarządzania użytkownikami. Pozwala on m. in.: na edycję uprawnień użytkowników, akceptację nowych użytkowników w systemie oraz dezaktywację kont. Rysunek A.6 przedstawia widok listy użytkowników. W celu aktualizacji uprawnień użytkownika w systemie należy kliknąć łącze „Edycja”.

Jesteś zalogowany jako **xh**. [Wyloguj się](#)

OCS - Ontology Creation System

[Strona Główna](#)

[O Nas](#)

[Pobieranie](#)

[Ontologie](#)

[Użytkownicy](#)

[Forum](#)

[English version](#) | [Wersja polska](#)

Lista użytkowników

Nazwa użyt.	Admin	Pozostało ont.	Pełna nazwa	E-mail	Zalogowany	Zaakceptowany	Edycja
xh	<input checked="" type="checkbox"/>	10	Lukasz Budnik	xh@vega.eti.pg.gda.pl	17.09.2008 19:02:26	<input checked="" type="checkbox"/>	Edycja
lukasz	<input checked="" type="checkbox"/>	10	Łukasz Budnik	lukasz.budnik@gmail.com	05.09.2008 12:01:47	<input checked="" type="checkbox"/>	Edycja
jakow	<input checked="" type="checkbox"/>	10	Andrzej Jakowski	j@wp.pl	17.09.2008 18:28:45	<input checked="" type="checkbox"/>	Edycja

© Copyright 2008

KASK

WETI

Politechnika Gdańska

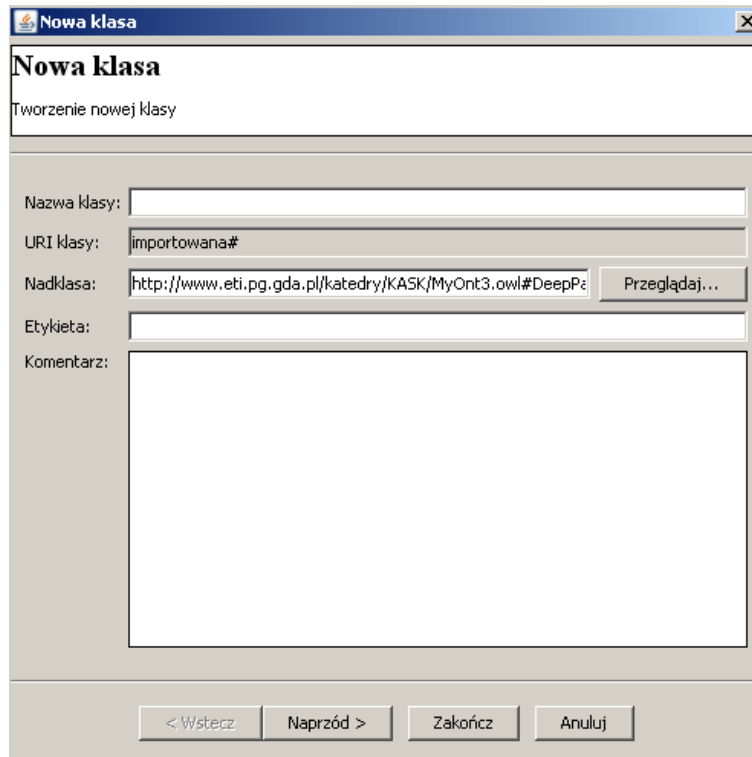
© Copyright 2008 KASK | WETI | Politechnika Gdańska

Rysunek A.6: Panel administracyjny (zarządzanie użytkownikami).

A.2 Edytor ontologiczny

Koncepcja budowy interfejsu graficznego edytora opiera się o „kreatory”. Pozwalają one na wywołanie większości funkcji w edytorze. Każdy kreator składa się z przynajmniej jednej strony, przycisków nawigacyjnych oraz panelu informacyjnego. W momencie wywołania dowolnego kreatora użytkownik proszony jest o podanie informacji wymaganych do wykonania danej funkcji programu. Po ich wprowadzeniu, wpisana treść jest walidowana

i zachowywana w systemie. Na rysunku A.7, przedstawiono widok kreatora tworzenia nowej podklasy.

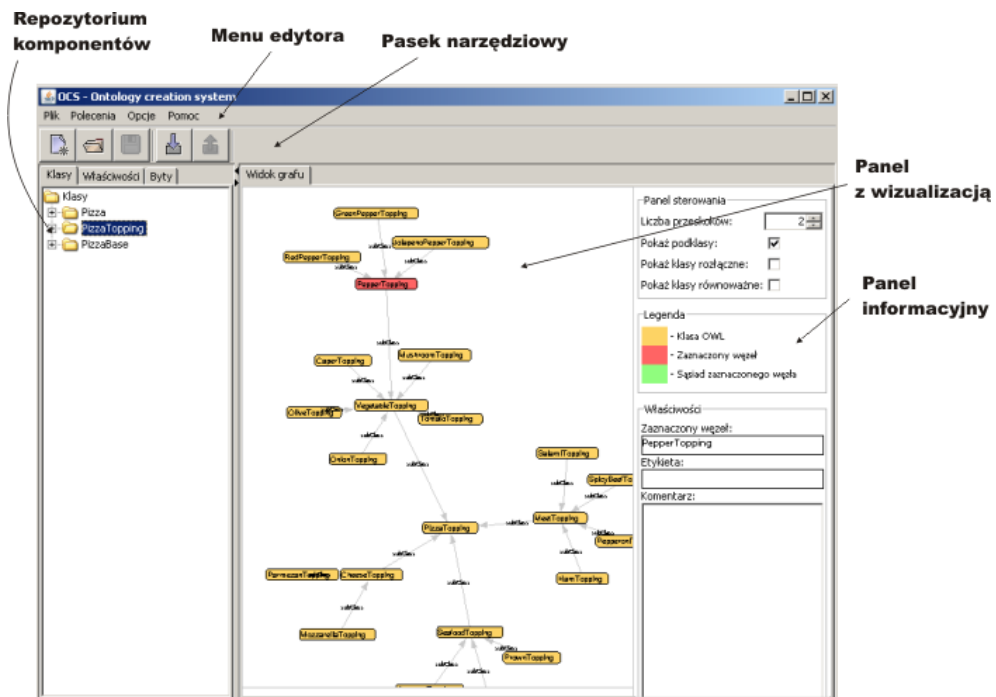


Rysunek A.7: Widok przykładowego kreatora dla edytora ontologicznego.

A.2.1 Widok edytora

Główny widok edytora przedstawionego na rysunku A.8, został podzielony na pięć elementów umożliwiających dostęp do funkcjonalności edycji ontologii i współpracy z częścią serwerową systemu OCS:

- **repozytorium komponentów** - odpowiedzialne jest za wyświetlanie elementów składowych ontologii: klas, właściwości i bytów. Z poziomu repozytorium możliwy jest dostęp do funkcjonalności edycji danego komponentu. Dokonuje się tego poprzez kliknięcie prawym klawiszem myszy na wybrany element ontologii. Zostanie wtedy rozwinięte menu kontekstowe z dostępnymi opcjami edycyjnymi,
- **menu edytora** - odpowiedzialne jest za dostęp do funkcji edytora, niezwiązanych bezpośrednio z edycją ontologii, lecz z zarządzaniem



Rysunek A.8: Struktura interfejsu edytora ontologicznego.

sesją edycji i współpracą z częścią serwerową systemu. Z poziomu menu możliwe jest pobranie ontologii, jej eksport, zatwierdzanie zmian oraz tworzenie nowej wersji ontologii,

- **pasek narzędziowy** - stanowi skrót do funkcji dostępnych w menu edytora,
- **panel z wizualizacją** - odpowiedzialny jest za wizualizację ontologii w postaci grafu. Węzłami grafu są klasy, natomiast krawędzie obrazują relacje pomiędzy klasami,
- **panel informacyjny** - wyświetla informacje na temat aktualnie zaznaczonego węzła na grafie. W panelu tym dostarczone są także możliwości dostosowania widoku grafu do potrzeb użytkownika: filtrowanie relacji, ustalenie maksymalnej liczby przeskoków.

A.2.2 Pobieranie ontologii

W celu pobrania ontologii z serwera systemu OCS, utworzony został specjalny kreator, który dostępny jest w pasku narzędziowym pod przyciskiem



Rysunek A.9: Funkcje edytora dostępne w pasku narzędziowym.

„Import ontologii”. W momencie jego wywołania, na pierwszej stronie kreatora, użytkownik proszony jest o wybór ontologii wraz z jej wersją. Po przejściu do następnej strony wymagane jest wprowadzenie nazwy projektu, w którym zostanie zapisana ontologia. Po wybraniu ontologii i podaniu nazwy projektu, ontologia zostanie ściągnięta z serwera OCS i zapisana lokalnie. W wyniku tych działań powstanie następująca struktura plików:

- plik *project.xml* - to tzw. deskryptor projektu, przechowuje on informacje odnośnie ściągniętej ontologii i jej wersji. Znajduje się w nim również informacja o lokalizacji pliku z ontologią pobraną z serwera systemu OCS (plik *base.owl*) oraz ontologią zmodyfikowaną,
- plik *base.owl* - to plik z ontologią pobraną z serwera systemu OCS. Plik ten nie powinien być modyfikowany, gdyż używany jest on przez edytor do pozyskiwania zmian wprowadzonych podczas edycji ontologii.

A.2.3 Ładowanie i edycja ontologii

Edytor ontologiczny umożliwia edycję ontologii pobranej z serwera systemu OCS, bądź z innego, dowolnego źródła. W celu otwarcia projektu, uprzednio pobranego z serwera OCS, należy wcisnąć przycisk „Otwórz” znajdujący się w pasku narzędziowym i wybrać dowolny deskryptor projektu (plik *project.xml*). Wtedy edytor załaduje odpowiednią ontologię z wprowadzonymi przedtem zmianami. Po załadowaniu projektu do edytora, istnieje możliwość wysłania propozycji zmian na serwer OCS (patrz punkt A.2.4).

W celu otwarcia dowolnej ontologii zapisanej w formacie OWL, bądź RDF, wystarczy wcisnąć przycisk „Otwórz” i wybrać interesujący nas plik. Ontologia zostanie załadowana do edytora. Istnieje możliwość edycji tej ontologii i zapisania jej ponownie na dysk. Nie można jednak wysyłać propozycji zmian, gdyż wymagane jest załadowanie projektu.

A.2.4 Przesyłanie zmian

Po otwarciu projektu i modyfikacji ontologii, możliwe jest przesłanie zmian na serwer OCS. W celu wyeksportowania zmian, należy wybrać opcję „Zapis zmian na serwerze”, znajdującą się w pasku narzędziowym. Należy wprowadzić informacje odnośnie tytułu zmian oraz ich opis. Informacje te wykorzystywane są przez właściciela ontologii podczas tworzenia nowej wersji ontologii.

A.2.5 Tworzenie nowej wersji

Każdy użytkownik ma możliwość tworzenia nowej wersji ontologii, utworzonej przez siebie, bądź tej, do której ma uprawnienia eksperckie. W celu tworzenia nowej wersji należy wybrać opcję „Utwórz nową wersję” z menu „Polecenia”. Powinien się pojawić kreator tworzenia nowej wersji. Na pierwszej stronie kreatora zostanie wyświetlona informacja o dostępnych ontologiach, na drugiej natomiast propozycje zmian zapisane na serwerze OCS. Właściciel, bądź ekspert ontologii, mogą wybrać propozycje zmian, które aprobują (chcą aby znalazły się w nowej wersji ontologii), bądź odrzucają (nie znajdują się one w nowej wersji ontologii).

A.2.6 Eksport ontologii

Edytor ontologiczny umożliwia także eksport aktualnie załadowanej ontologii do pliku. W tym celu należy wybrać „Eksport” z menu „Plik” i wybrać format zapisu ontologii. W pierwszej wersji edytor obsługuje format RDF i OWL.

Dodatek B

Szczegółowy opis klas

B.1 Klasa *MainFrame*

Opis: Główna klasa systemu odpowiedzialna za tworzenie widoku edytora. Składa się z kilku paneli, które stanowią poszczególne fragmenty widoku edytora.

Atrybuty: Brak.

Metody:

- *initialize()* - metoda inicjalizująca poszczególne komponenty głównej ramki edytora.

B.2 Klasa *Logic*

Opis: Klasa realizująca wzorzec *singleton*. Odpowiedzialna jest za przeprowadzanie całego przetwarzania związanego z wywoływaniem metod biznesowych systemu OCS. Zawiera również metody narzędziowe, ułatwiające współpracę z biblioteką OWL API.

Atrybuty: Brak.

Metody:

- *createSubClassRelation()* - metoda tworząca aksjomat nadklasa-podklasa dla zadanych identyfikatorów URI dwóch klas,
- *createLabelAxiom()* - metoda tworząca aksjomat „etykiety”, który może być dołączony do dowolnego elementu ontologii,

- *getOntologiesVersions()* - metoda pobierająca informacje o wersjach i podwersjach dla wszystkich ontologii w systemie,
- *loginUser()* - metoda logująca użytkownika w systemie,
- *createDeleteClassAxiom()* - metoda zwracająca wszystkie aksjomaty powiązane z klasą przeznaczoną do usunięcia,
- *applyChanges()* - metoda zatwierdzająca wszystkie zmiany wprowadzone do ontologii,
- *displayReferencedClasses()* - metoda wyświetlająca wszystkie klasy w ontologii.

B.3 Klasa *AbstractWizard*

Opis: Klasa abstrakcyjna reprezentująca podstawowy kreator w edytorze ontologicznym. Każdy utworzony w systemie kreator musi dziedziczyć od tej klasy. Zawiera ona szereg metod pozwalających na tworzenie i rozbudowę wszelkiego rodzaju kreatorów. Każdy kreator składa się z kilku stron, które są instancjami klasy *AbstractPage*.

Atrybuty:

- *AbstractPage pages[]* - atrybut zawierający wszystkie strony kreatora,
- *AbstractPage currentPage* - atrybut określający aktualnie wyświetlaną stronę kreatora.

Metody:

- *doStart()* - metoda wywoływana w momencie rozpoczęcia działania kreatora. Wykonuje wstępne przetwarzanie związane z inicjalizacją kreatora,
- *doFinish()* - metoda wywoływana po przejściu walidacji wszystkich stron kreatora, w momencie naciśnięcia przycisku „Zakończ”,
- *doCancel()* - metoda wywoływana w momencie anulowania działania kreatora, w momencie naciśnięcia przycisku „Anuluj”,
- *getStartPage()* - metoda zwracająca pierwszą stronę kreatora, która jest wyświetlana w momencie wywołania kreatora.

B.4 Klasa *AbstractPage*

Opis: Klasa abstrakcyjna reprezentująca pojedynczą stronę kreatora. Strona każdego kreatora powinna dziedziczyć od tej klasy. Zawiera ona podstawowe metody pozwalające na walidację, jak również nawigację do kolejnych stron kreatora.

Atrybuty: Brak.

Metody:

- *getContainer()* - metoda zwracająca kontener z elementami graficznymi, które mają zostać wyświetlone na danej stronie kreatora,
- *validate()* - metoda walidująca zawartość pól znajdujących się na danej stronie kreatora.

B.5 Klasa *GraphPanel*

Opis: Klasa odpowiedzialna za tworzenie widoku ontologii w postaci grafu. Posiada metody aktualizujące odpowiednie elementy grafu, jak również przenoszące fokus na konkretne węzły grafu.

Atrybuty: Brak.

Metody:

- *initialize()* - metoda inicjalizująca poszczególne komponenty graficzne znajdujące się w tym panelu,
- *updatePropertiesPanel()* - metoda aktualizująca zawartość panelu z właściwościami na podstawie wartości obecnie zaznaczonej klasy,
- *updateUriLabel()* - metoda aktualizująca etykietę z URI na podstawie wartości obecnie zaznaczonej klasy.

B.6 Klasa *GraphDisplay*

Opis: Klasa pośrednicząca pomiędzy klasą *GraphPanel*, a klasami komponentu do wizualizacji grafów - *prefuse* (6.4). Inicjalizuje widok grafu, odpowiada za jego aktualizację i dostarcza odpowiednich metod pozwalających na jego filtrowanie.

Atrybuty:

- *Visualization visualization* - referencja do klasy z pakietu *prefuse* w obrębie którego, realizowana jest wizualizacja grafu,
- *Graph graph* - obiekt reprezentujący graf, który ma zostać wyświetlony w ramach konkretnej wizualizacji,
- *Filter graphDistanceFilter* - filtr pozwalający na wyświetlanie określonych węzłów grafu, oddalonych od danego węzła o zadaną liczbę przeskoków,
- *Filter relationsFilter* - filtr pozwalający na wyświetlanie części powiązań w danym grafie. Filtr może posłużyć do pominięcia w wyświetlaniu elementów, które posiadają pewne powiązania z innymi elementami w grafie, np.: filtrowanie relacji nadklasa-podklasa, bądź relacji rozłączności klas.

Metody:

- *focusNodeOnGraph()* - metoda przenosząca fokus na dany węzeł grafu,
- *initializeVisualization()* - metoda inicjalizująca komponent odpowiedzialny za przeprowadzenie wizualizacji ontologii,
- *initializeDisplay()* - metoda inicjalizująca obiekt widoku danej wizualizacji ontologii.

B.7 Klasa *TreePanel*

Opis: Klasa odpowiedzialna za tworzenie widoku ontologii w postaci drzewa. Posiada metody aktualizujące odpowiednie elementy drzewa, jak również przenoszące fokus na konkretne jego węzły.

Atrybuty: Brak.

Metody:

- *initialize()* - metoda inicjalizująca dany komponent, tworząca wszystkie elementy widoku i rozmieszczająca je w obrębie danego panelu,
- *updateURIlabel()* - metoda aktualizująca zawartość pola z nazwą aktualnie zaznaczonego elementu ontologii na drzewie,
- *updateCommentLabel()* - metoda aktualizująca zawartość pola z komentarzem aktualnie zaznaczonego elementu ontologii na drzewie.

B.8 Klasa *ComponentRepositoryPanel*

Opis: Klasa odpowiedzialna za wyświetlanie poszczególnych elementów ontologii tak, aby były one dostępne do bezpośredniej edycji. Komponenty te wyświetlane są w postaci drzewiastej i zostały podzielone na następujące elementy: *klasy*, *właściwości* oraz *byty*.

Atrybuty:

- *String selectedClass* - pole przechowujące informacje o aktualnie zaznaczonej klasie,
- *String selectedProperty* - pole przechowujące informacje o aktualnie zaznaczonej właściwości,
- *String selectedIndividual* - pole przechowujące informacje o aktualnie zaznaczonym bycie,

Metody:

- *displayOntologyClasses()* - metoda wyświetlająca klasy danej ontologii w obrębie tego panelu,
- *displayOntologyIndividuals()* - metoda wyświetlająca byty danej ontologii w obrębie tego panelu,
- *displayOntologyProperties()* - metoda wyświetlająca właściwości danej ontologii w obrębie tego panelu,

Spis symboli i skrótów

Skrót	Opis
ABox	Assertional Box - pudełko asercjonalne, jeden z podstawowych elementów ontologii wyrażonych w językach logiki opisowej.
API	Application Programming Interface - interfejs programowania aplikacji, specyfikacja procedur i funkcji zewnętrznych w stosunku do aplikacji z niego korzystającej.
DAML	DARPA Agent Markup Language - wczesny język reprezentacji ontologii.
DL	Description Logic - logika opisowa.
EJB	Enterprise Java Beans - specyfikacja dotycząca komponentów serwera aplikacji wykonanych w technologii <i>Java</i> .
FOL	First Order Logic - rachunek predykatów pierwszego rzędu.
GUI	Graphical User Interface - graficzny interfejs użytkownika.
HTML	HyperText Markup Language - hipertekstowy język znaczników, dominujący język wykorzystywany do tworzenia stron internetowych.
HTTP	HyperText Transfer Protocol - protokół przesyłania dokumentów hipertekstowych.
HTTPS	HyperText Transfer Protocol Secure - bezpieczny protokół przesyłania dokumentów hipertekstowych.
JEE	Java Enterprise Edition - wydanie <i>Javy</i> , przeznaczone do tworzenia aplikacji wykorzystywanych w przemyśle.
JIT	Just In Time Compiler - mechanizm optymalizacji kodu klas <i>Javy</i> .

JNDI	Java Naming and Directory Interface - interfejs <i>Javy</i> dla usług katalogowych, pozwalający na odkrywanie i wyszukiwanie danych oraz obiektów za pomocą nazw.
JNLP	Java Network Launching Protocol - protokół wdrażania aplikacji <i>Javy</i> , przy wykorzystaniu technologii <i>Java Web Start</i> .
JSF	Java Server Faces - specyfikacja <i>Javy</i> pozwalająca na tworzenie aplikacji, wykorzystująca wzorzec MVC.
JSP	Java Server Pages - technologia <i>Javy</i> , pozwalająca na tworzenie dynamicznych stron internetowych.
MVC	Model View Controller - wzorzec projektowy pozwalający na odseparowanie widoku od modelu danych i akcji operujących na tych danych.
OCS	Ontology Creation System - system tworzenia i składowania ontologii.
OIL	Ontology Inference Layer - wczesny język zapisu ontologii.
OWL	Web Ontology Language - rodzina języków zapisu ontologii.
RDF	Resource Description Framework - model opisu zasobów.
RDFS	Resource Description Framework Schema - schemat RDF, definiujący jego podstawowe konstrukcje.
SSO	Single Sign-On - wzorzec projektowy, umożliwiający przeprowadzenie pojedynczego procesu uwierzytelnienia.
TBox	Terminological Box - pudełko terminologiczne, jeden z podstawowych elementów ontologii wyrażonych w językach logiki opisowej
UML	Unified Modeling Language - ujednolicony język modelowania.
URI	Uniform Resource Identifier - standard internetowy pozwalający na identyfikację zasobów w sieci.
W3C	The World Wide Web Consortium - konsorcjum zajmujące się tworzeniem standardów internetowych.
WWW	World Wide Web - system informacyjny oparty na publicznie dostępnych.
XML	eXtensible Markup Language - rozszerzalny język znaczników, uniwersalny język formalny przeznaczony do reprezentowania różnych danych w ustrukturalizowany sposób.

Spis rysunków

2.1	Przykładowa ontologia wyrażona za pomocą ramek Minsky’ego.	11
2.2	Hierarchia dla ontologii ekosystemu.	15
2.3	Struktura logiczna Semantycznej Sieci Web (diagram pochodzi ze strony www.semanticweb.org).	16
2.4	Graf RDF przedstawiający opis strony WWW.	18
2.5	Drzewo genealogiczne rodziny języków OWL.	20
3.1	Zrzut ekranu z przykładowego edytora (tutaj Protégé).	30
3.2	Przykładowe drzewo jakości.	32
3.3	Drzewo jakości zdefiniowane w celu oceny edytorów ontologicznych.	39
4.1	Sposoby wykorzystania systemu OCS.	47
4.2	Architektura logiczna systemu OCS.	50
4.3	Proces wprowadzania zmian do ontologii.	53
4.4	Ontologia po wprowadzeniu zmian.	54
4.5	Edycja ontologii - widok z poziomu użytkownika.	54
5.1	Diagram przypadków użycia dla systemu OCS.	59
5.2	Diagram klas dla edytora ontologicznego.	66
6.1	Uproszczony model OWL API [4].	72
6.2	Struktura pliku wdrożeniowego technologii Java Web Start. . . .	77
6.3	Architektura silnika <i>prefuse</i> [1].	80
6.4	Zrzut ekranu edytora ontologicznego z przykładową ontologią. .	84
7.1	Wykres średniego czasu serializacji do postaci trójkowej.	92
7.2	Wykres średniego czasu pobierania ontologii z serwera OCS. . .	94
7.3	Wykres średniego czasu przesyłania ontologii na serwer OCS. . .	95

A.1	Główny szablon strony.	104
A.2	Formularz logowania.	104
A.3	Formularz rejestracji nowego użytkownika.	105
A.4	Pobieranie edytora ontologicznego.	106
A.5	Zarządzanie ontologiami.	106
A.6	Panel administracyjny (zarządzanie użytkownikami).	107
A.7	Widok przykładowego kreatora dla edytora ontologicznego. . . .	108
A.8	Struktura interfejsu edytora ontologicznego.	109
A.9	Funkcje edytora dostępne w pasku narzędziowym.	110

Spis tabel

2.1	Przykładowe języki logik opisowych.	13
2.2	Przykładowa ontologia prostego ekosystemu, bazująca na ontologii <i>rodziny</i> z [10].	14
5.1	Struktura pliku konfiguracyjnego.	68
5.2	Struktura deskryptora projektu.	69
7.1	Przeprowadzone testy funkcjonalne.	89
7.2	Ontologie użyte w testach.	91
7.3	Czas serializacji ontologii do postaci trójkowej.	92
7.4	Rozmiary komunikatów i średni czas pobierania ontologii z serwera OCS.	93
7.5	Rozmiary komunikatów oraz średni czas przesyłania na serwer OCS.	94